US Flight data analysis - Part 1

'Data Analysis with Python' by David Taieb. Published by Packt Publishing, 2018:
https://learning.oreilly.com/library/view/data-analysis-with/9781789950069/ch09s03.html (https://learning.oreilly.com/library/view/data-analysis-with/9781789950069/ch09s03.html)

1. Load data into Pandas DataFrames using pixiedust.sampleData()
2. Create a networkx DiGraph from the flights DataFrame
3. Augment the airports DataFrame with columns that compute the different centrality indices
4. Visualize airports using networkx drawing APIs and pixiedust display() Mapbox Map
5. Compute the Dikjstra shorted path between 2 airports using ELAPSED_TIME and the centrality indices as weight

Pixiedust

is an open source Python helper library that works as an add-on to Jupyter notebooks to improve the user experience of working with data.

Homepage: https://pixiedust.github.io/pixiedust/index.html (https://pixiedust.github.io/pixiedust/index.html)

To install:

#!pip install pixiedust

In [2]:

```
import pixiedust
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import Image
import matplotlib.cm as cm
```

```
Pixiedust database opened successfully
```

 (https://github.com/ibm-watson-data-lab/pixiedust) Pixiedust version 1.1.17

Warning: You are not running the latest version of PixieDust. Current is 1.1.17, Latest is 1.1.18

Please copy and run the following command in a new cell to upgrade:
`!pip install --user --upgrade pixiedust`

Please restart kernel after upgrading.

Load Airports

List of all U.S. airports including their IATA code (International Air Transport Association), city, state, longitude, and latitude.

Airports Data: https://openflights.org/data.html (https://openflights.org/data.html)

In [2]:

```
airports = pixiedust.sampleData("https://github.com/DTAIEB/Thoughtf
```

```
Downloading 'https://github.com/DTAIEB/Thoughtful-Data
-Science/raw/master/chapter%209/USFlightsAnalysis/airp
orts.csv' from https://github.com/DTAIEB/Thoughtful-Da
ta-Science/raw/master/chapter%209/USFlightsAnalysis/ai
rports.csv (https://github.com/DTAIEB/Thoughtful-Data-
Science/raw/master/chapter%209/USFlightsAnalysis/airpo
rts.csv)
Downloaded 23867 bytes
Creating pandas DataFrame for 'https://github.com/DTAI
EB/Thoughtful-Data-Science/raw/master/chapter%209/USFl
ightsAnalysis/airports.csv'. Please wait...
Loading file using 'pandas'
Successfully created pandas DataFrame for 'https://git
hub.com/DTAIEB/Thoughtful-Data-Science/raw/master/chap
ter%209/USFlightsAnalysis/airports.csv'
```

In [3]:

```
##download as file
display(airports)
```

Schema

Table

Search table

Showing 100 of 322 rows

IATA_CODE	AIRPORT	CITY

Load airlines - list of U.S. airlines including their IATA code.

In [4]:

```
airlines = pixiedust.sampleData("https://github.com/DTAIEB/Thoughtf
```

```
Downloading 'https://github.com/DTAIEB/Thoughtful-Data
-Science/raw/master/chapter%209/USFlightsAnalysis/airl
ines.csv' from https://github.com/DTAIEB/Thoughtful-Da
ta-Science/raw/master/chapter%209/USFlightsAnalysis/ai
rlines.csv (https://github.com/DTAIEB/Thoughtful-Data-
Science/raw/master/chapter%209/USFlightsAnalysis/airli
nes.csv)
Downloaded 359 bytes
Creating pandas DataFrame for 'https://github.com/DTAI
EB/Thoughtful-Data-Science/raw/master/chapter%209/USFl
ightsAnalysis/airlines.csv'. Please wait...
Loading file using 'pandas'
Successfully created pandas DataFrame for 'https://git
hub.com/DTAIEB/Thoughtful-Data-Science/raw/master/chap
ter%209/USFlightsAnalysis/airlines.csv'
```

In [5]:

```
airlines.head(10)
```

Out[5]:

	IATA_CODE	AIRLINE
0	UA	United Air Lines Inc.
1	AA	American Airlines Inc.
2	US	US Airways Inc.
3	F9	Frontier Airlines Inc.
4	B6	JetBlue Airways
5	OO	Skywest Airlines Inc.
6	AS	Alaska Airlines Inc.
7	NK	Spirit Air Lines
8	WN	Southwest Airlines Co.
9	DL	Delta Air Lines Inc.

Load flights - list of flights that occurred in 2015. This data includes date, origin and destination airports, scheduled and actual times, and delays.

In [6]:

```
flights = pixiedust.sampleData("https://github.com/DTAIEB/Thoughtfu
```

```
Downloading 'https://github.com/DTAIEB/Thoughtful-Data
-Science/raw/master/chapter%209/USFlightsAnalysis/flig
hts.zip' from https://github.com/DTAIEB/Thoughtful-Dat
a-Science/raw/master/chapter%209/USFlightsAnalysis/fli
ghts.zip (https://github.com/DTAIEB/Thoughtful-Data-Sc
ience/raw/master/chapter%209/USFlightsAnalysis/flights
.zip)
Extracting first item in zip file...
File extracted: flights.csv
Downloaded 86144241 bytes
Creating pandas DataFrame for 'https://github.com/DTAI
EB/Thoughtful-Data-Science/raw/master/chapter%209/USFl
ightsAnalysis/flights.zip'. Please wait...
Loading file using 'pandas'
Successfully created pandas DataFrame for 'https://git
hub.com/DTAIEB/Thoughtful-Data-Science/raw/master/chap
ter%209/USFlightsAnalysis/flights.zip'
```

In [31]:

```
flights.head()
```

Out[31]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	ORIG
0	2015	1	1	4	AS	98	
1	2015	1	1	4	AA	2336	
2	2015	1	1	4	US	840	
3	2015	1	1	4	AA	258	
4	2015	1	1	4	AS	135	

Load the data into a networkx directed weighted graph object using the flights DataFrame as the edge list and the values from the ELAPSED_TIME column as the weight.

In [8]:

```
edges = flights.groupby(["ORIGIN_AIRPORT", "DESTINATION_AIRPORT"])[
edges
```

Out[8]:

		ELAPSED_TIME
ORIGIN_AIRPORT	**DESTINATION_AIRPORT**	
ABE	**ATL**	127.415350
	DTW	101.923741
	ORD	130.298762
ABI	**DFW**	53.951591
ABQ	**ATL**	174.822278
...	**...**	...
XNA	**SFO**	249.901961
	SLC	155.000000
YAK	**CDV**	49.193846
	JNU	45.956923
YUM	**PHX**	57.270227

4656 rows × 1 columns

Reset the index from a multi-index to a regular single index converting the index columns into regular columns.

In [9]:

```
edges = edges.reset_index()
edges.head(10)
```

Out[9]:

	ORIGIN_AIRPORT	DESTINATION_AIRPORT	ELAPSED_TIME
0	ABE	ATL	127.415350
1	ABE	DTW	101.923741
2	ABE	ORD	130.298762
3	ABI	DFW	53.951591
4	ABQ	ATL	174.822278
5	ABQ	BWI	215.028112
6	ABQ	CLT	193.168421
7	ABQ	DAL	95.107051
8	ABQ	DEN	75.268199
9	ABQ	DFW	103.641714

To create the directed weighted graph, we use the NetworkX ***from_pandas_edgelist()*** method which takes a pandas DataFrame as the input source. We also specify the source and target columns, as well as the weight column (in our case ELAPSED_TIME). Finally, we tell NetworkX that we want to create a directed graph by using the create_using keyword arguments, passing an instance of DiGraph as a value.

Weighted graph example: [https://qxf2.com/blog/drawing-weighted-graphs-with-networkx/ (https://qxf2.com/blog/drawing-weighted-graphs-with-networkx/)](https://qxf2.com/blog/drawing-weighted-graphs-with-networkx/)

In [10]:

```
flight_graph = nx.from_pandas_edgelist(
    df=flights,
    source="ORIGIN_AIRPORT",
    target="DESTINATION_AIRPORT",
    edge_attr="ELAPSED_TIME", # weight
    create_using = nx.DiGraph() )
```

We can quickly validate that our graph has the right values by directly printing its nodes and edges:

In [11]:

```
print("Nodes: {}".format(flight_graph.nodes))
```

```
Nodes: ['ANC', 'SEA', 'LAX', 'PBI', 'SFO', 'CLT', 'MIA
', 'MSP', 'LAS', 'DFW', 'ATL', 'DEN', 'SLC', 'IAH', 'P
DX', 'MCI', 'FAI', 'FLL', 'PHX', 'ORD', 'HNL', 'SJU',
'EWR', 'JFK', 'PBG', 'IAG', 'PSE', 'MCO', 'BQN', 'BOS'
, 'BDL', 'GEG', 'ITO', 'ONT', 'KOA', 'OGG', 'MYR', 'HI
B', 'ABR', 'MKE', 'BNA', 'DTW', 'BRO', 'VPS', 'BOI', '
BJI', 'LIH', 'PHL', 'SBN', 'EUG', 'IAD', 'BUF', 'PWM',
'CRP', 'PIA', 'FAT', 'SMF', 'AUS', 'BWI', 'JAX', 'MFR'
, 'IDA', 'MSN', 'DCA', 'SAT', 'CHS', 'SBA', 'IND', 'CL
E', 'GSP', 'RIC', 'BFL', 'OMA', 'RDM', 'CID', 'TPA', '
SYR', 'ROC', 'TYR', 'LAN', 'GSO', 'LGA', 'RSW', 'OAK',
'PVD', 'RNO', 'PIT', 'ABQ', 'HOU', 'TUL', 'LIT', 'MSY'
, 'OKC', 'SJC', 'LGB', 'ATW', 'PNS', 'MEM', 'TYS', 'MH
T', 'SAV', 'GRB', 'ABE', 'JAN', 'OAJ', 'FAR', 'ERI', '
LEX', 'CWA', 'TTN', 'RDU', 'CVG', 'BHM', 'ACY', 'RAP',
'TUS', 'EAU', 'DLH', 'FSD', 'INL', 'BRD', 'SPI', 'CLD'
, 'COD', 'CMH', 'PSC', 'CPR', 'ACV', 'DAL', 'PAH', 'MR
Y', 'ESC', 'ISN', 'PSP', 'CAE', 'STL', 'BTV', 'MTJ', '
GCC', 'RKS', 'MBS', 'GUC', 'ORF', 'MOT', 'MLU', 'SNA',
'MOB', 'SAN', 'LAW', 'PIB', 'MEI', 'MGM', 'SBP', 'COS'
, 'LAR', 'DRO', 'BIS', 'MDW', 'BTR', 'HLN', 'BZN', 'MD
T', 'SCE', 'TWF', 'OTZ', 'BPT', 'GPT', 'STC', 'HPN', '
MLB', 'PLN', 'CIU', 'CAK', 'DSM', 'BLI', 'SHV', 'BUR',
'ALB', 'LNK', 'CMI', 'GTF', 'EKO', 'AVL', 'HSV', 'XNA'
, 'SUX', 'HYS', 'MFE', 'ISP', 'DAB', 'DAY', 'LFT', 'LB
E', 'ASE', 'ELP', 'GUM', 'SCC', 'TVC', 'ALO', 'TLH', '
STT', 'MHK', 'IMT', 'RHI', 'JNU', 'KTN', 'JAC', 'DBQ',
```

```
'GNV', 'DIK', 'SDF', 'LBB', 'AVP', 'SGF', 'COU', 'BTM'
, 'SRQ', 'ELM', 'PIH', 'SUN', 'LWS', 'GJT', 'VEL', 'PU
B', 'HRL', 'HDN', 'ORH', 'SAF', 'YUM', 'FCA', 'GRR', '
BIL', 'ROA', 'CHA', 'EYW', 'CRW', 'MQT', 'CHO', 'CDC',
'EGE', 'FWA', 'APN', 'ECP', 'MMH', 'CEC', 'EVV', 'MSO'
, 'AMA', 'MLI', 'GFK', 'GRK', 'AGS', 'LRD', 'STX', 'IC
T', 'ILM', 'WRG', 'FNT', 'CNY', 'BRW', 'MKG', 'FLG', '
JMS', 'OME', 'GGG', 'LSE', 'ROW', 'ACT', 'SIT', 'FSM',
'CMX', 'FAY', 'MAF', 'AEX', 'BMI', 'DHN', 'PSG', 'TOL'
, 'SGU', 'PHF', 'BET', 'AZO', 'CLL', 'RST', 'TRI', 'VL
D', 'BQK', 'SWF', 'CSG', 'EWN', 'GCK', 'SMX', 'RDD', '
LCH', 'GTR', 'DVL', 'ILG', 'ABY', 'ADK', 'GRI', 'UST',
'JLN', 'TXK', 'SPS', 'ABI', 'YAK', 'SJT', 'CDV', 'OTH'
, 'ADQ', 'PPG', 'HOB', 'BGM', 'BGR', 'ITH', 'ACK', 'MV
Y', 'WYS', 'DLG', 'AKN', 'GST', 'HYA']
```

In [12]:

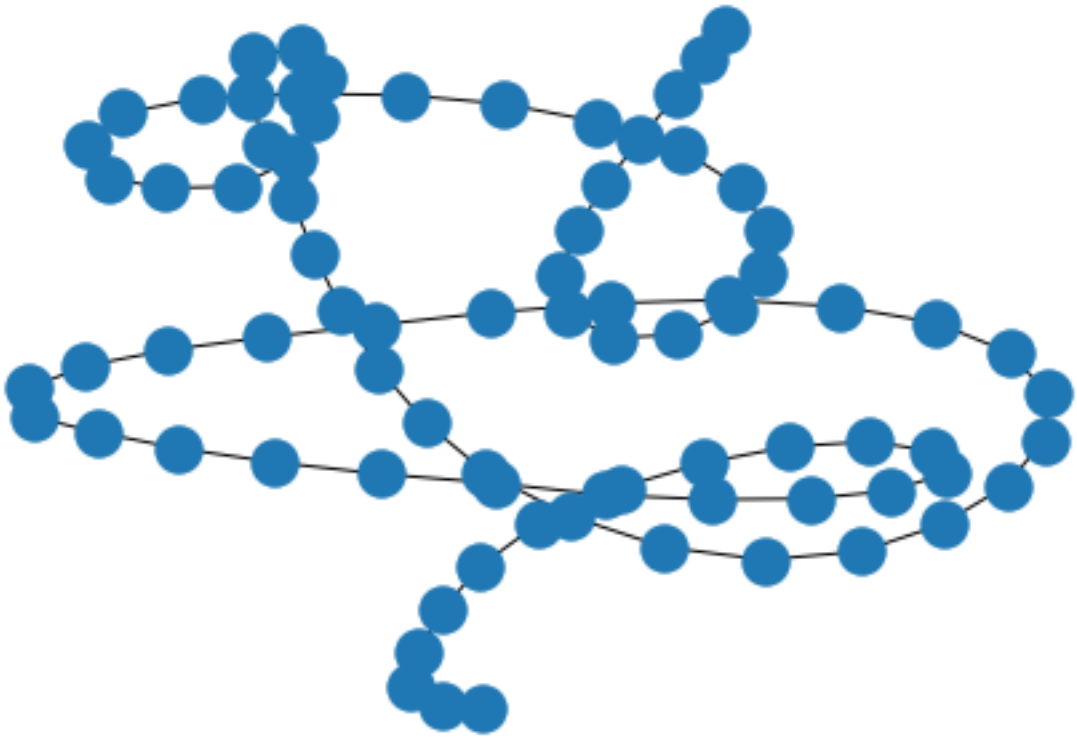
```
print("Edges: {}".format(flight_graph.edges))
```

```
Edges: [('ANC', 'SEA'), ('ANC', 'PDX'), ('ANC', 'PHX')
, ('ANC', 'MSP'), ('ANC', 'OTZ'), ('ANC', 'SCC'), ('AN
C', 'JNU'), ('ANC', 'OGG'), ('ANC', 'OME'), ('ANC', 'B
ET'), ('ANC', 'HNL'), ('ANC', 'ADK'), ('ANC', 'SFO'),
('ANC', 'ORD'), ('ANC', 'LAS'), ('ANC', 'FAI'), ('ANC'
, 'LAX'), ('ANC', 'DEN'), ('ANC', 'KOA'), ('ANC', 'ADQ
'), ('ANC', 'CDV'), ('ANC', 'BRW'), ('ANC', 'IAH'), ('
ANC', 'LGB'), ('ANC', 'ATL'), ('ANC', 'SLC'), ('ANC',
'DFW'), ('ANC', 'DLG'), ('ANC', 'AKN'), ('ANC', 'EWR')
, ('SEA', 'ANC'), ('SEA', 'MSP'), ('SEA', 'MIA'), ('SE
A', 'PHX'), ('SEA', 'DEN'), ('SEA', 'IAH'), ('SEA', 'D
FW'), ('SEA', 'EWR'), ('SEA', 'SJC'), ('SEA', 'OAK'),
('SEA', 'SFO'), ('SEA', 'LAS'), ('SEA', 'IAD'), ('SEA'
, 'LAX'), ('SEA', 'SNA'), ('SEA', 'ORD'), ('SEA', 'MDW
'), ('SEA', 'ATL'), ('SEA', 'PHL'), ('SEA', 'SAN'), ('
SEA', 'SLC'), ('SEA', 'PSP'), ('SEA', 'SMF'), ('SEA',
'DTW'), ('SEA', 'PDX'), ('SEA', 'JFK'), ('SEA', 'ONT')
, ('SEA', 'JNU'), ('SEA', 'KTN'), ('SEA', 'BUR'), ('SE
A', 'GEG'), ('SEA', 'MCO'), ('SEA', 'DCA'), ('SEA', 'B
WI'), ('SEA', 'HNL'), ('SEA', 'TPA'), ('SEA', 'LIH'),
```

Default graph layout called **spring_layout**, which is a force-directed layout. The nodes in the graph are positioned as if they are connected by springs and their final positions determined by a minimum of stretching of the edges.

In [13]:

```
G=nx.path_graph(80)
pos=nx.spring_layout(G)
nx.draw(G)
```



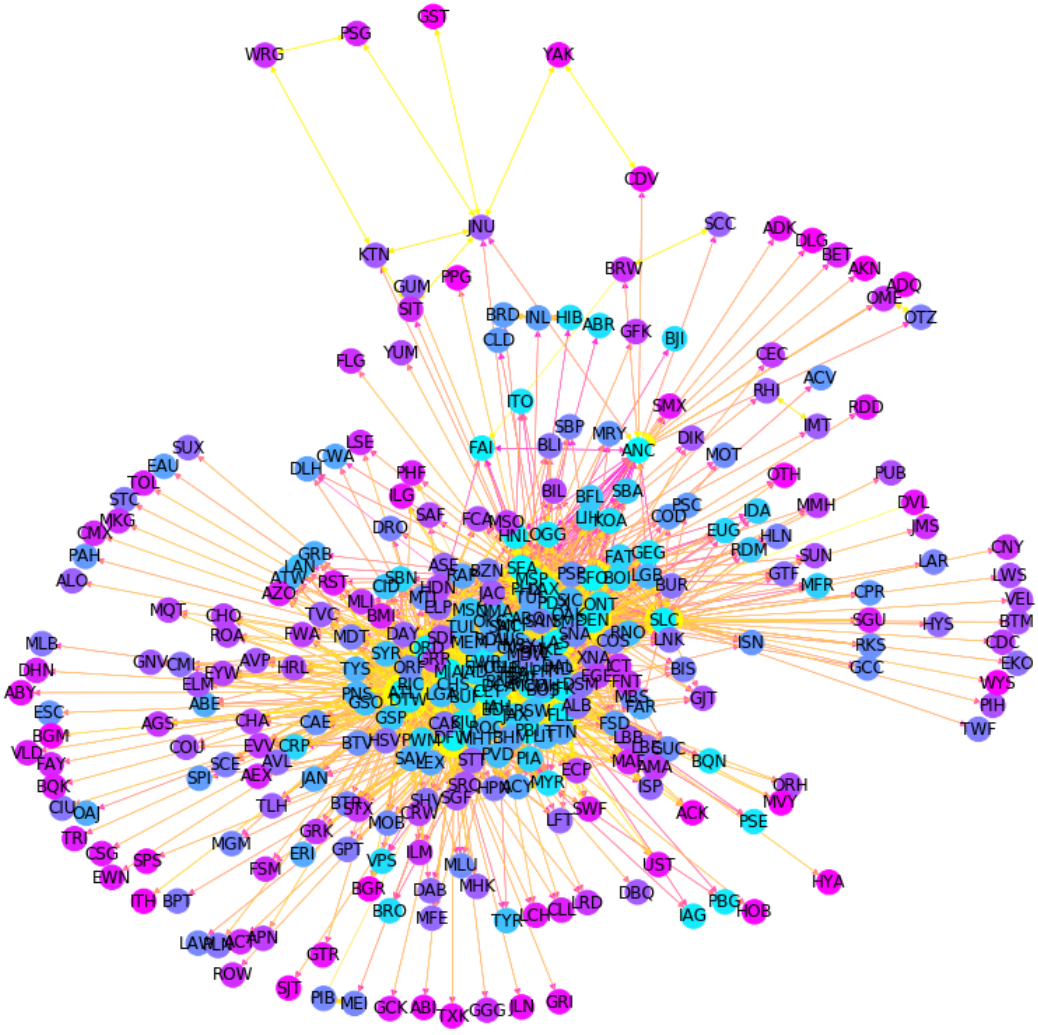
The NetworkX ***draw()*** method uses Matplotlib engine.

To beautify the visualization:

- we configure it with proper width and height (12, 12);
- add a colormap with vivid color (we use the cool and spring colormap from matplolib.cm, see: https://matplotlib.org/2.0.2/examples/color/colormaps_reference.html (https://matplotlib.org/2.0.2/examples/color/colormaps_reference.html)).

In [14]:

```
fig = plt.figure(figsize = (12,12))

nx.draw(flight_graph, arrows=True, with_labels=True,
        width = 0.5,style="dotted",
        node_color=range(len(flight_graph)),
        cmap=cm.get_cmap(name="cool"),
        edge_color=range(len(flight_graph.edges)),
        edge_cmap=cm.get_cmap(name="spring")
       )
plt.show()
```



Colormaps

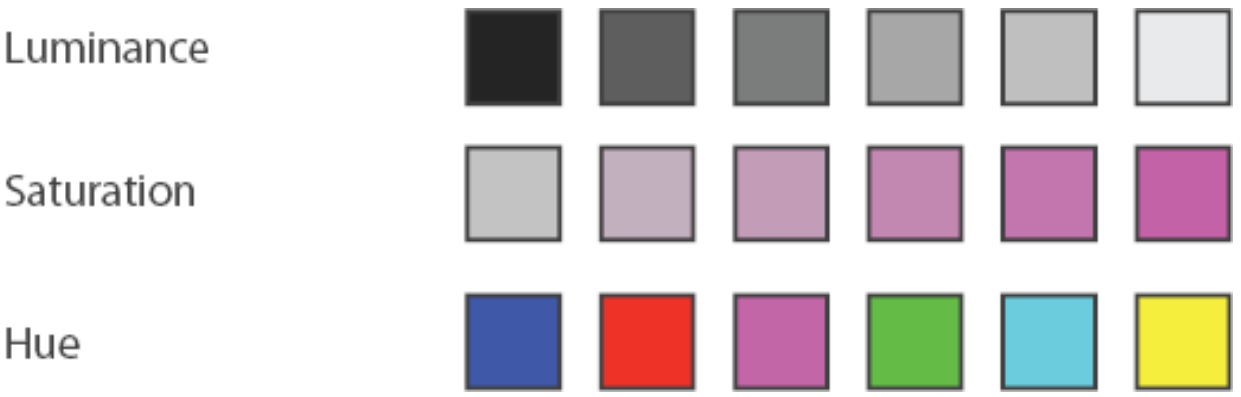
'*Visualization Analysis and Design*' by Tamara Munzner, Published by A K Peters/CRC Press, 2014. Chapter 10: https://learning.oreilly.com/library/view/visualization-analysis-and/9781466508910/K14708_C010.xhtml (https://learning.oreilly.com/library/view/visualization-analysis-and/9781466508910/K14708_C010.xhtml)

The colloquial term **color** is best understood in terms of three separate channels: **luminance, saturation and hue**.

In [4]:

```
Image('img/US_color.png')
```

Out[4]:



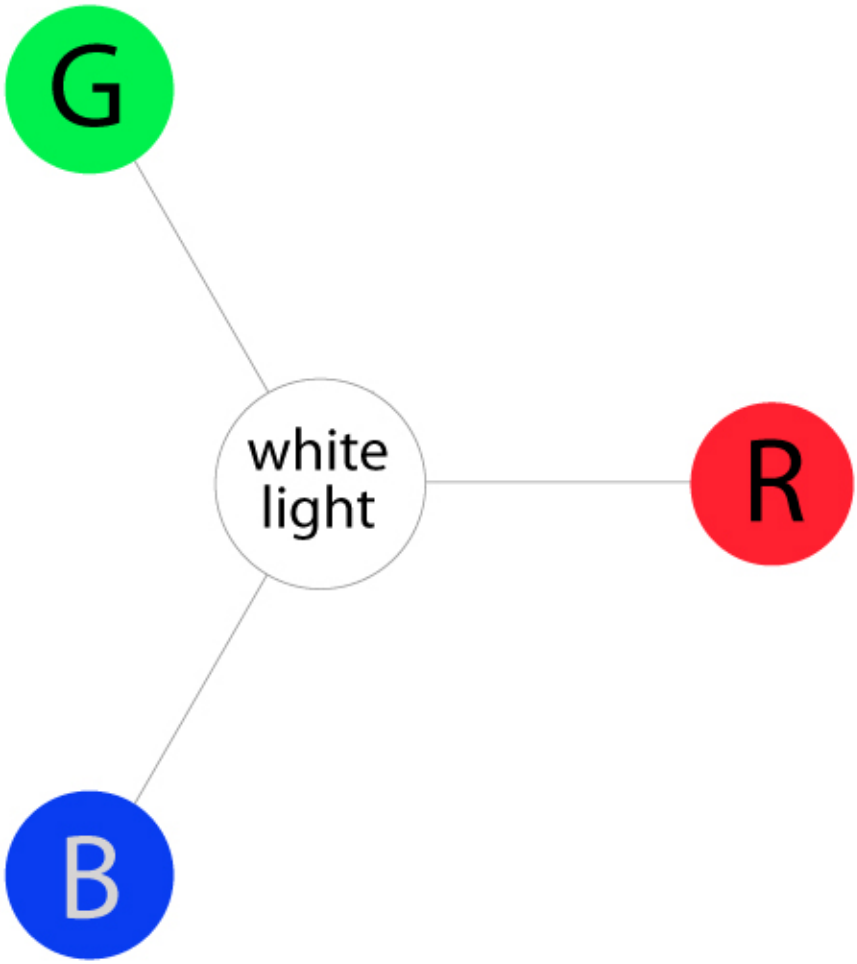
Hue

The three **primary hues** in light are red, green, and blue (RGB system). In light, all three of these wavelengths added together at full strength produces pure white light. The absence of all three of these colors produces complete darkness, or black.
http://learn.leighcotnoir.com/artspeak/elements-color/hue-value-saturation/ (http://learn.leighcotnoir.com/artspeak/elements-color/hue-value-saturation/)

In [5]:

```
Image('img/US_Primary_hue.jpg')
```

Out[5]:



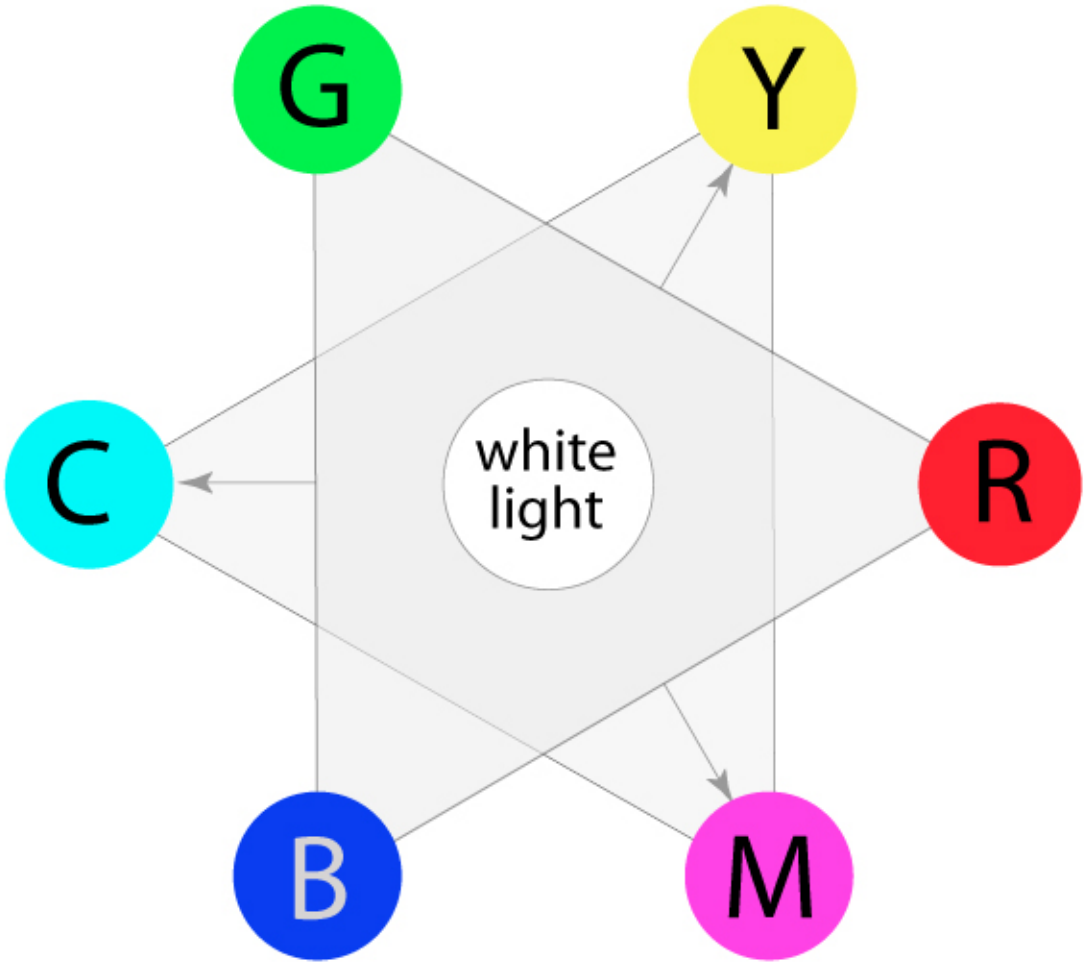
Mixing adjacent primaries will produce **secondary hues**:

Blue + Red light –> Magenta
Red + Green light –> Yellow
Green + Blue light –> Cyan

In [6]:

```
#Primary and Secondary hues
Image('img/US_Secondary_hues.jpg')
```

Out[6]:

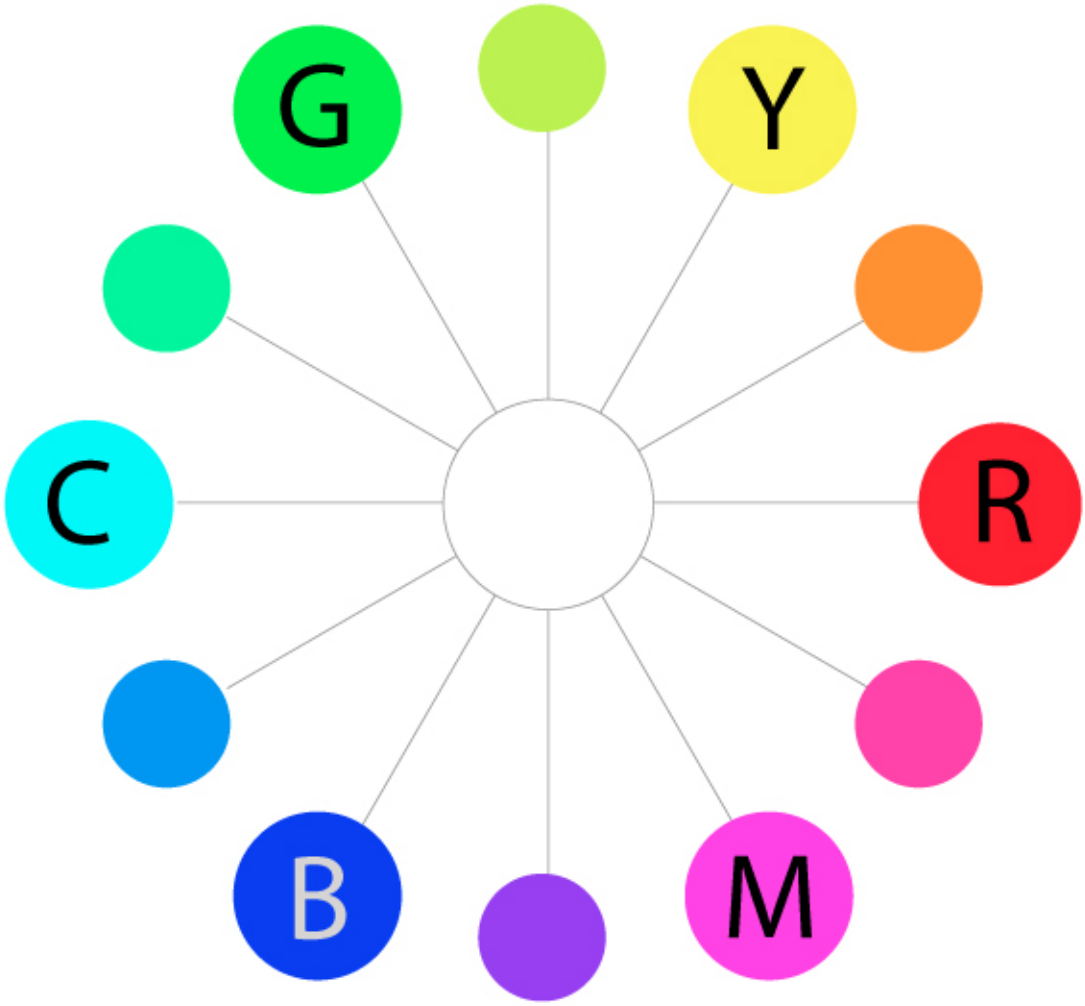


The colors on the outermost perimeter of the color circle are the"hues," which are colors in their purest form. This process can continue filling in colors around the wheel. The next level colors, **the tertiary colors**, are those colors between the secondary and primary colors.

In [10]:

```
Image('img/US_tertiary_hues.jpg')
#Primary, Secondary and Tertiary hues
```

Out[10]:



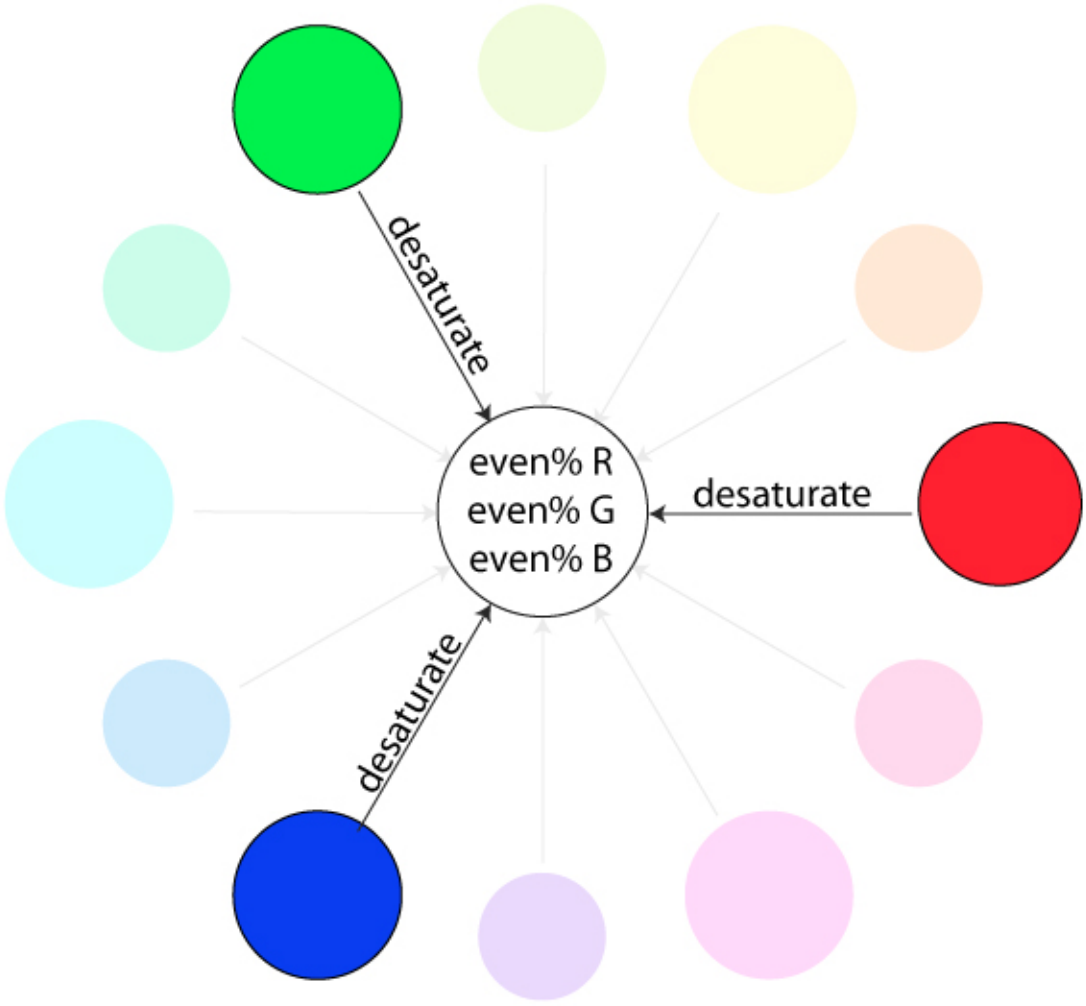
Saturation

Saturation is also referred to as “intensity”. It refers to the dominance of hue in the color. On the outer edge of the hue wheel are the ‘pure’ hues. As you move into the center of the wheel, the hue we are using to describe the color dominates less and less. When you reach the center of the wheel, no hue dominates. These colors directly on the central axis are considered desaturated.

In [11]:

```
Image('img/US_desaturate.jpg')
```

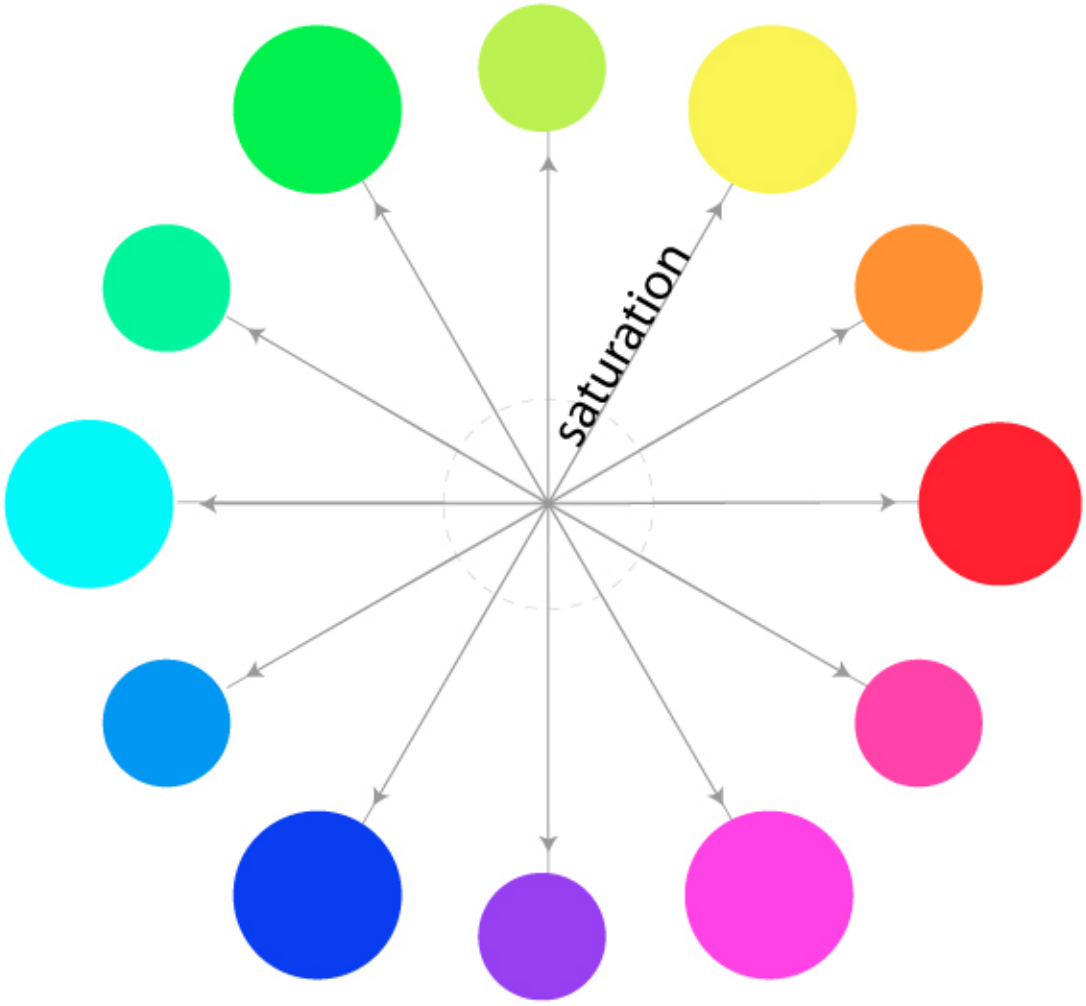
Out[11]:



In [14]:

```
Image('img/US_Saturate.jpg')
```

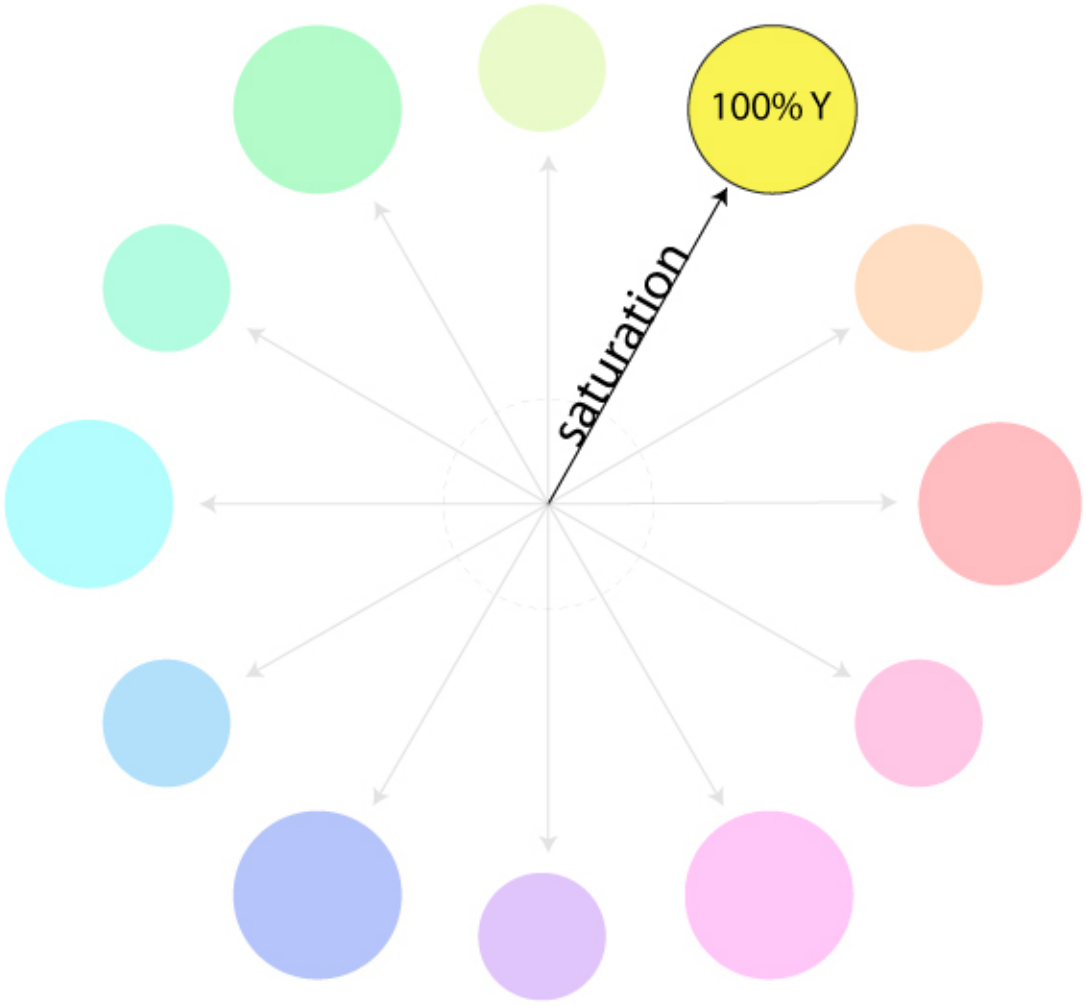
Out[14]:



In [15]:

```
Image('img/US_Complete_Saturation.jpg')
```

Out[15]:



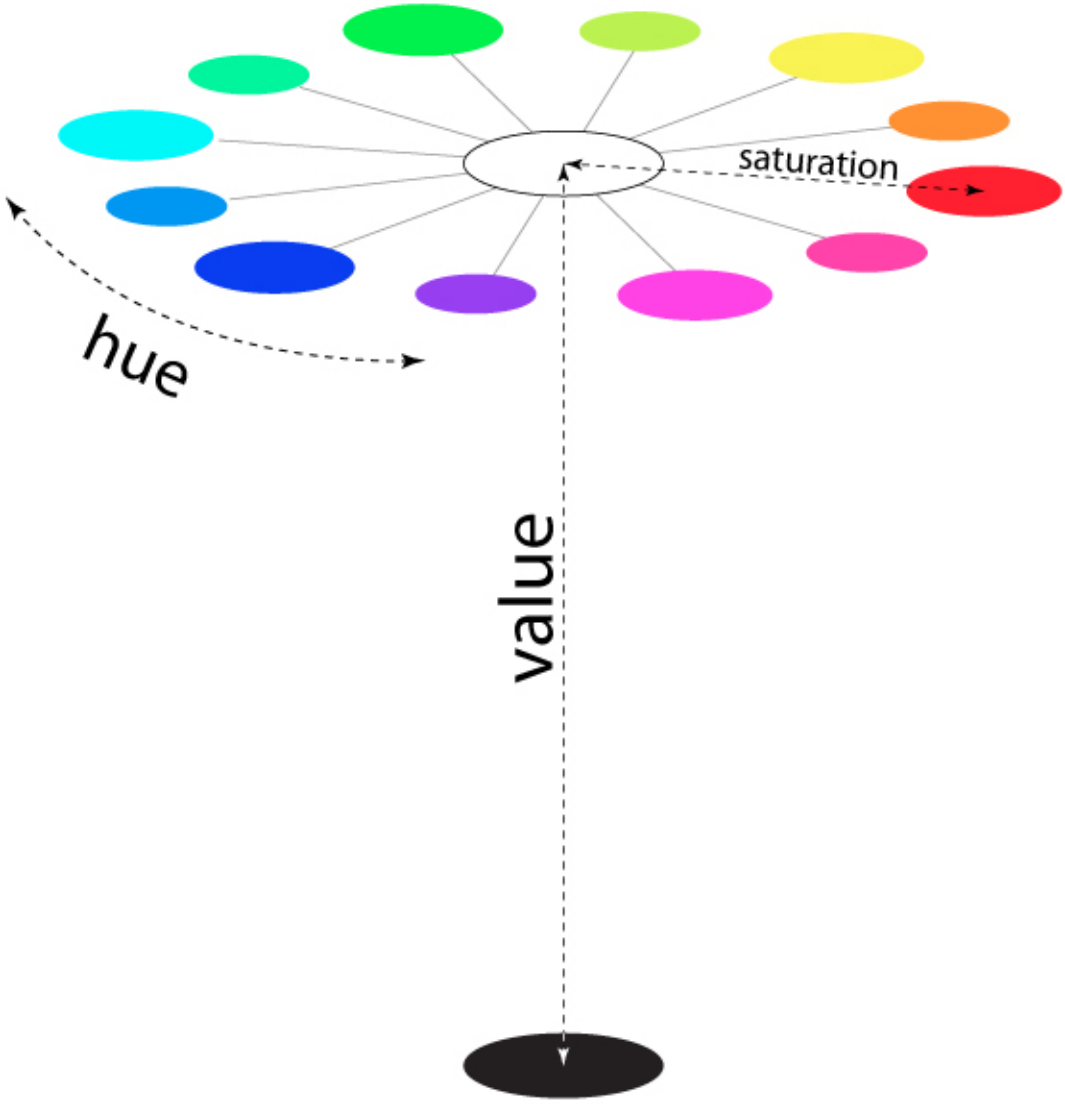
Value in HSV Scale

Value is the dimension of lightness / darkness. It describes the overall intensity or strength of the light. If hue can be thought of as a dimension going around a wheel, then value is a linear axis running through the middle of the wheel, as seen below:

In [17]:

```
#HSV model with Hue, Saturation and Value explained.
Image('img/US_HSV.jpg')
```

Out[17]:

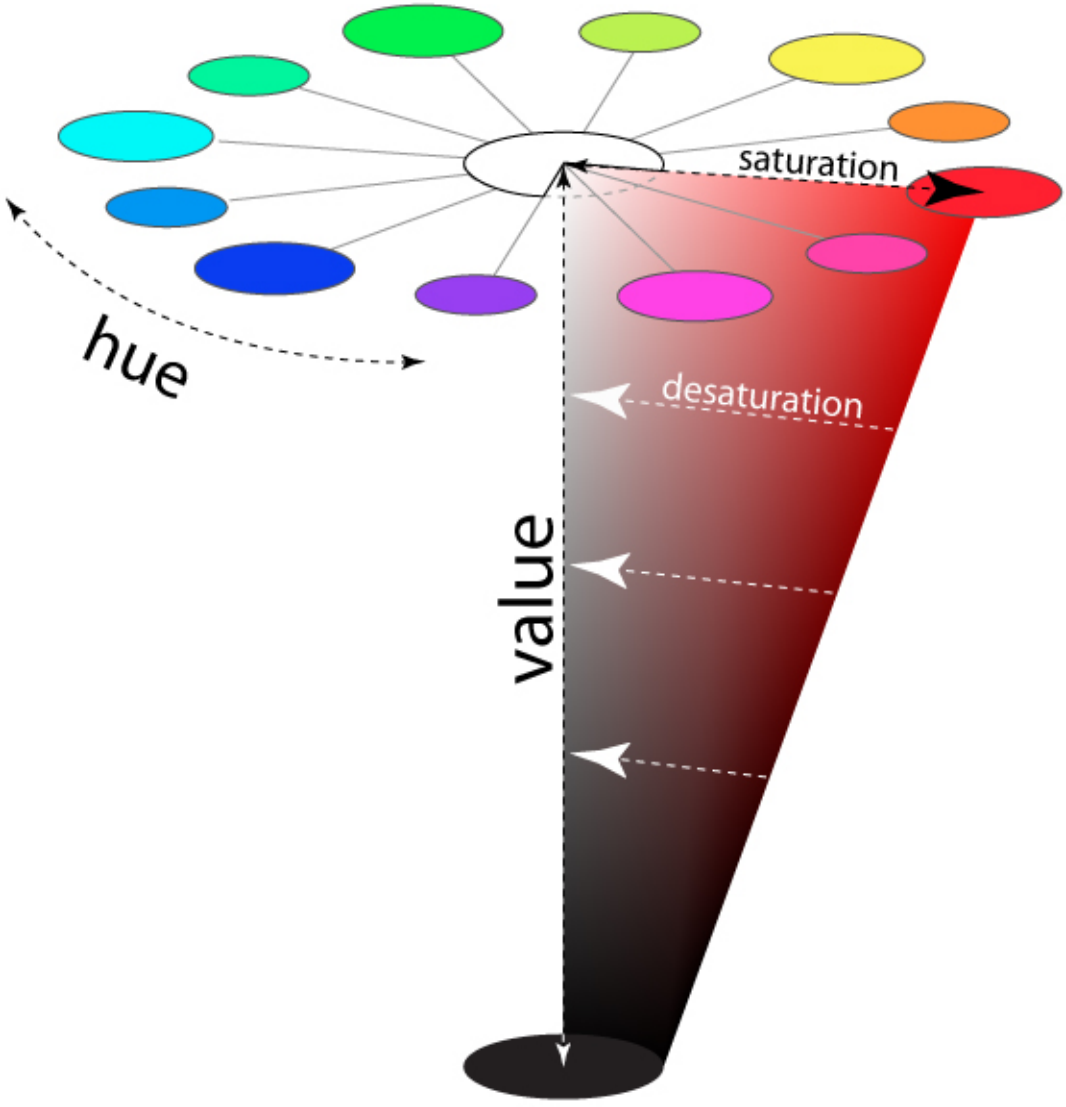


Example below shows a full color range for a single hue.

In [3]:

```
#HSV Model With Full Range of Single Hue
Image('img/US_HSV_single_hue.jpg')
```

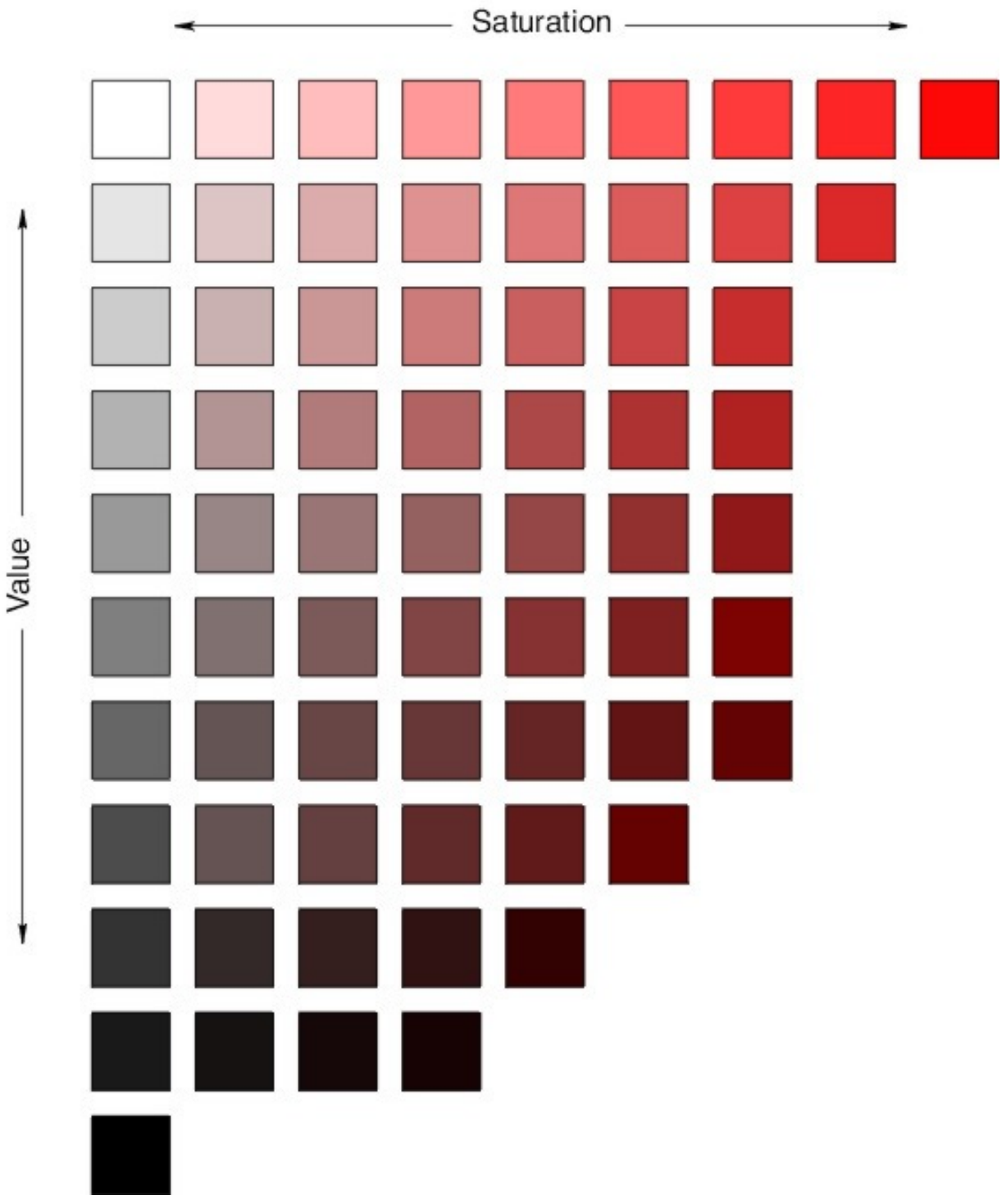
Out[3]:



In [5]:

```
Image('img/US_cone_slice.jpg')
```

Out[5]:



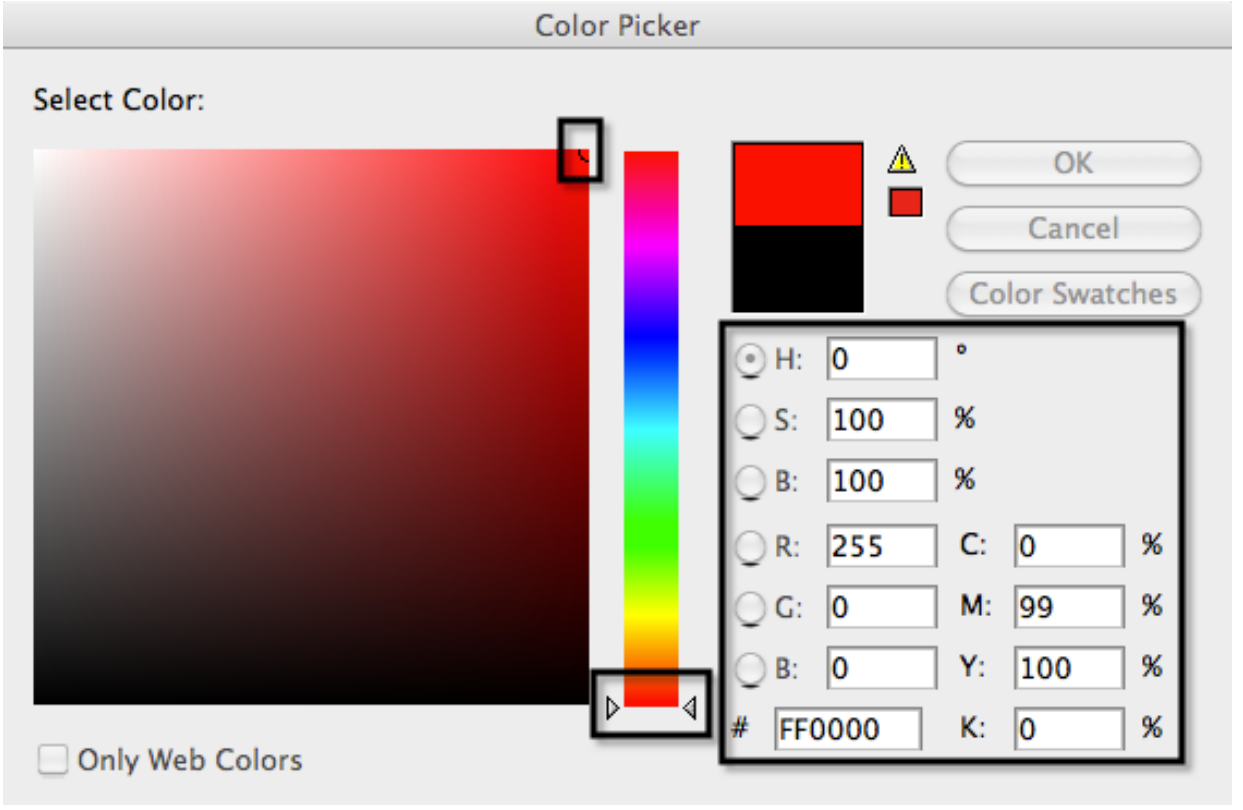
Color pickers

The center vertical slider is where we select the hue. It is currently set to the lowest selection and corresponds to the “H:0” radio button value on the right. The “H” indicates “Hue,” and the zero value describes which numerical hue assignment we have selected. Below it, you will see that “Red” is set to “255,” or the fullest level of light represented on a computer (0 = lowest). Notice that Blue and Green are set to zero, indicating that Red is at its fullest level of saturation.

In [6]:

```
Image('img/US_pick_pure_red_hue.png')
```

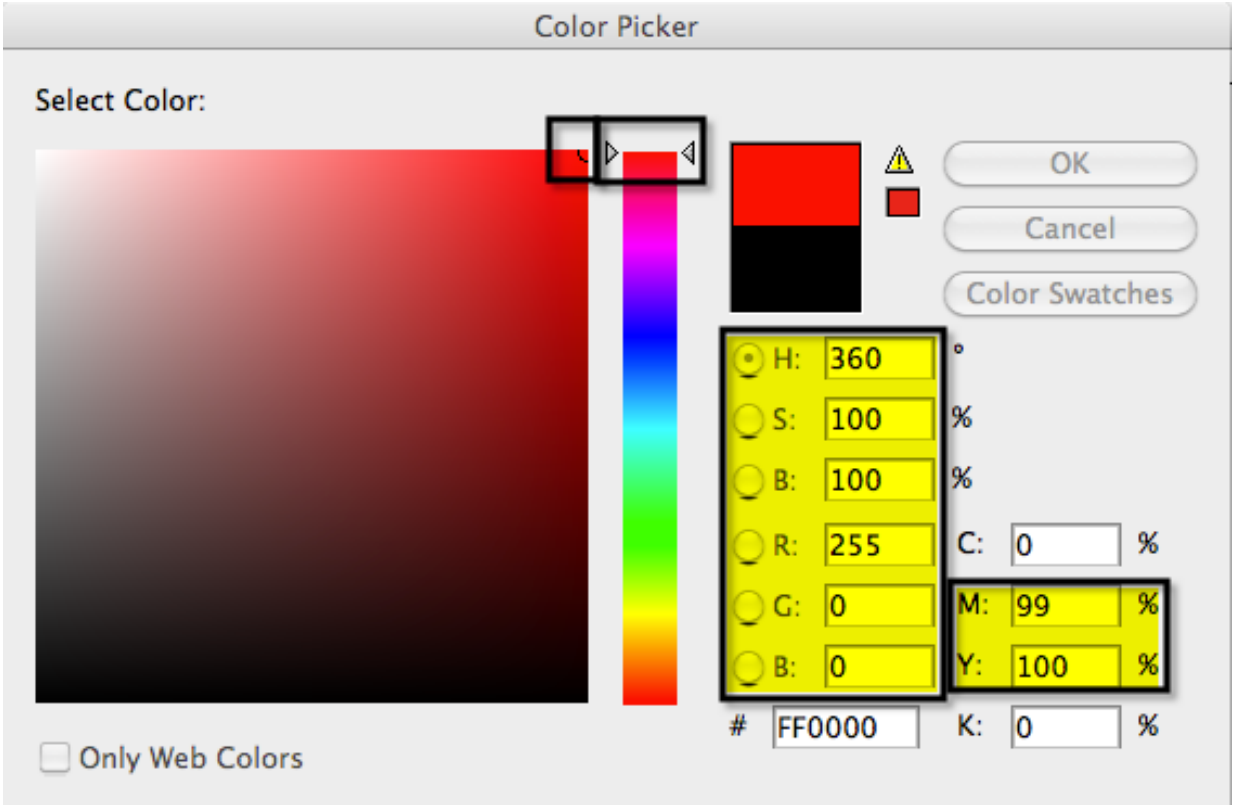
Out[6]:



In [7]:

```
Image('img/US_pick_pure_red_hue2.png')
```

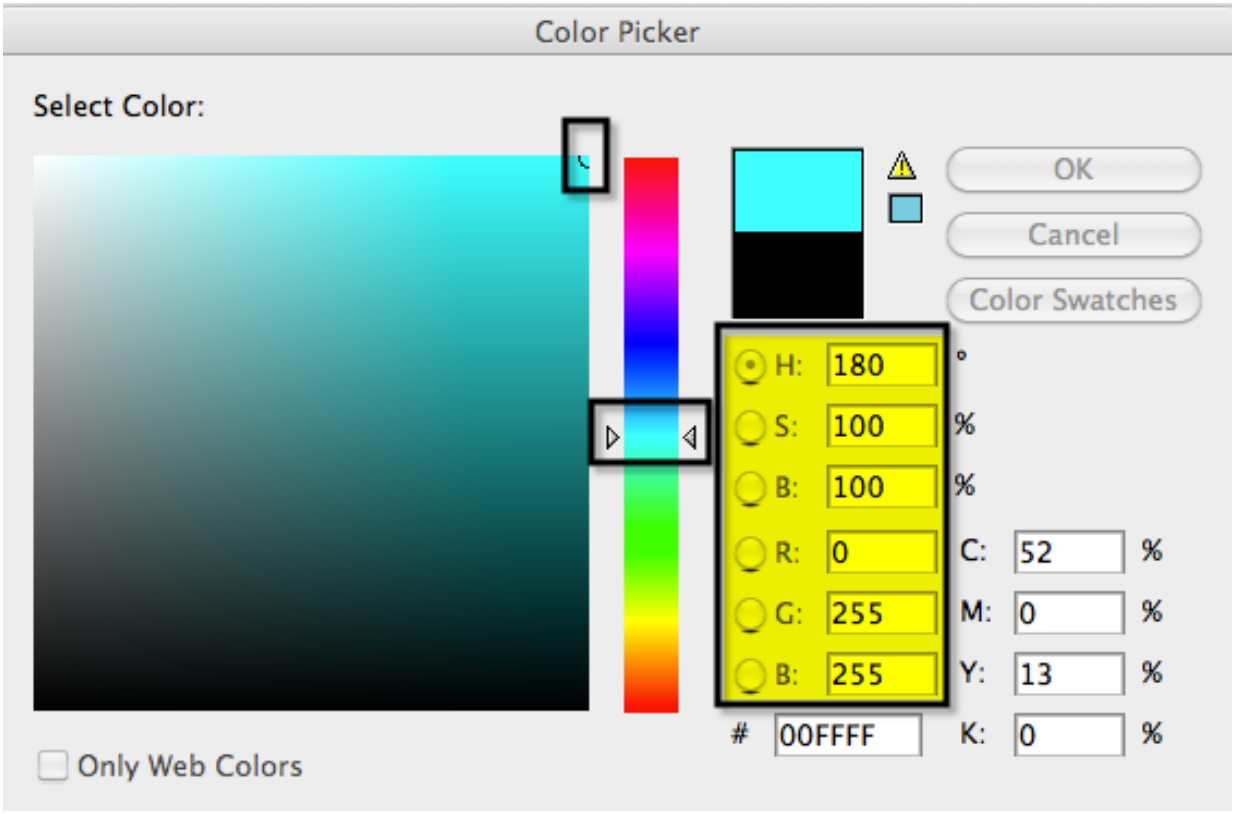
Out[7]:



In [8]:

```
Image('img/US_pick_cyan.png')
```

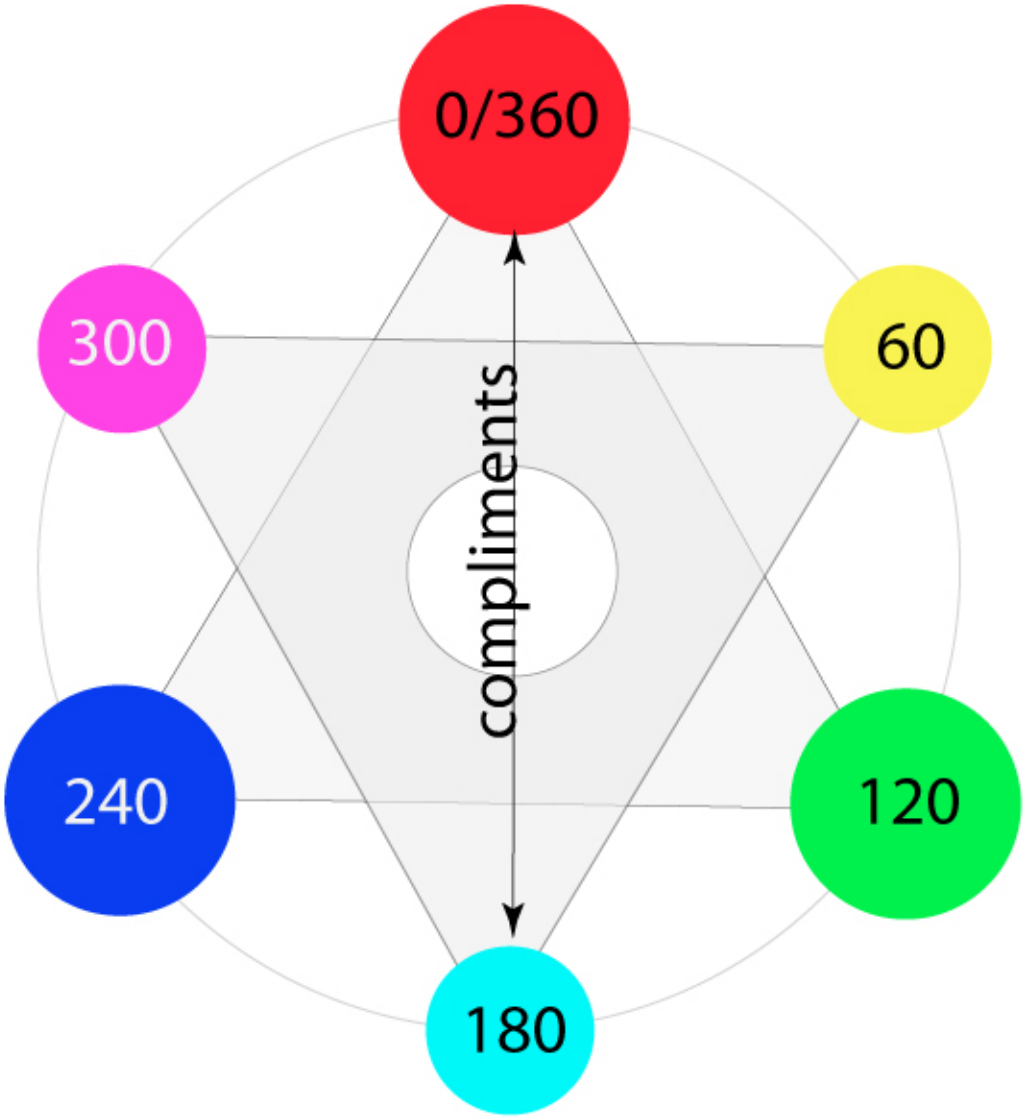
Out[8]:



In [11]:

```
Image('img/US_hue_degrees.jpg')
```

Out[11]:



A colormap specifies a mapping between colors and data values; that is, a visual encoding with color.

Node Colormap:
https://networkx.github.io/documentation/stable/auto_examples/drawing/plot_node_colorm
(https://networkx.github.io/documentation/stable/auto_examples/drawing/plot_node_colorm

Edge Colormap:
https://networkx.github.io/documentation/stable/auto_examples/drawing/plot_edge_colorm
(https://networkx.github.io/documentation/stable/auto_examples/drawing/plot_edge_colorm

Colormap reference:
https://matplotlib.org/2.0.2/examples/color/colormaps_reference.html
(https://matplotlib.org/2.0.2/examples/color/colormaps_reference.html)

Map a continuous vs categorical variable to node: https://python-graph-gallery.com/324-map-a-color-to-network-nodes/ (https://python-graph-gallery.com/324-map-a-color-to-network-nodes/)

In a **cycle** or circular graph, vertices are connected in a closed chain.

A **spring layout** is a Fruchterman-Reingold force-directed layout algorithm. The idea is to consider a force between any two nodes. In this algorithm, the nodes are represented by steel rings and the edges are springs between them. The **attractive force** is analogous to the spring force and the **repulsive force** is analogous to the electrical force. The basic idea is to minimize the energy of the system by moving the nodes and changing the forces between them.

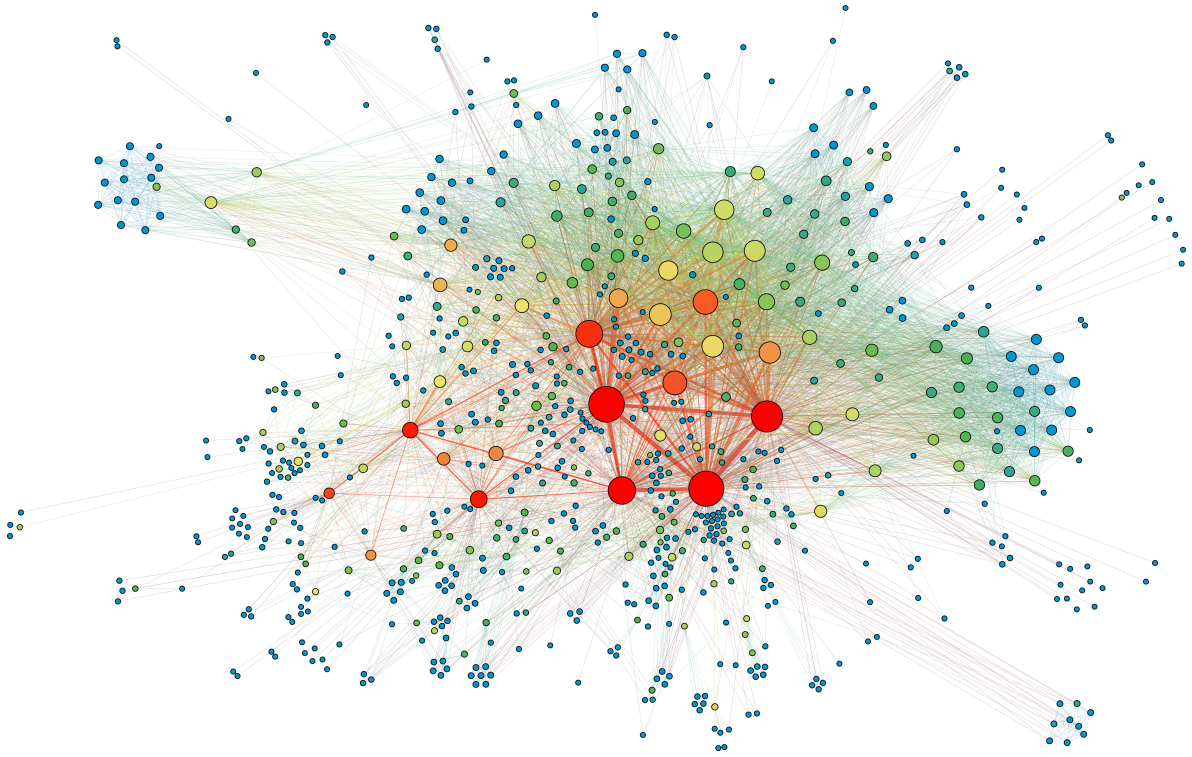
Social network visualization using a force-directed graph drawing algorithm.

Graph contains 800 vertices and 10,000 edges. Color = Betweenness centrality.
https://en.wikipedia.org/wiki/Force-directed_graph_drawing
(https://en.wikipedia.org/wiki/Force-directed_graph_drawing)

In [15]:

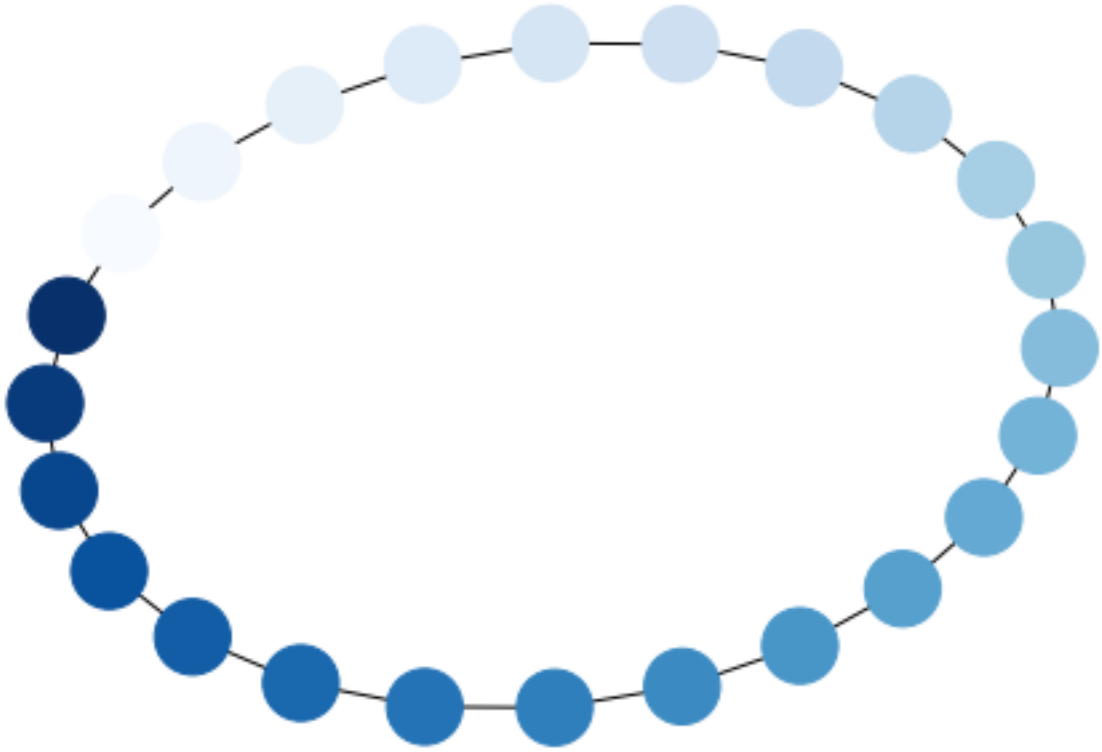
```
Image("US_SocialNetworkAnalysis.png")
```

Out[15]:



In [16]:

```
G = nx.cycle_graph(24)
pos = nx.spring_layout(G, iterations=500)
nx.draw(G, pos, node_color=range(24), node_size=800, cmap=plt.cm.Bl
plt.show()
```



In [17]:

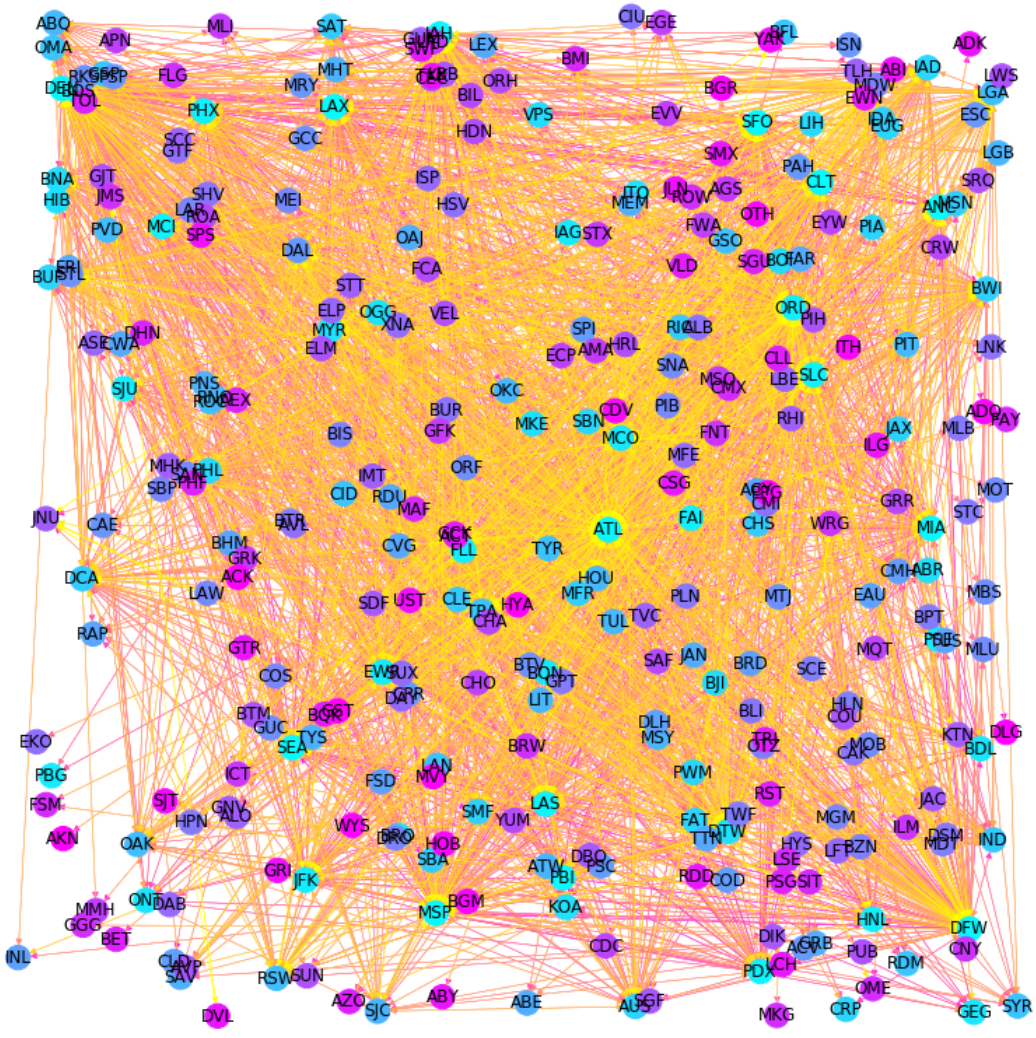
```
G = nx.star_graph(20)
pos = nx.spring_layout(G)

nx.draw(G, pos, node_color='#A0CBE2', edge_color=range(20),
        width=4, edge_cmap=plt.cm.Blues, with_labels=False) #width
plt.show()
nx.draw_networkx?
```

One benefit of the spring layout is that it quickly reveals the nodes with the most edge connections which are located at the center of the graph.

networkx supports other types of layout including circular_layout, random_layout, shell_layout, and spectral_layout.

In [18]:

```
fig = plt.figure(figsize = (12,12))
nx.draw(flight_graph,
        arrows=True,
        with_labels=True,
        width = 0.5,
        style="dotted",
        node_color=range(len(flight_graph)),
        cmap=cm.get_cmap(name="cool"),
        edge_color=range(len(flight_graph.edges)),
        edge_cmap=cm.get_cmap(name="spring"),
        pos = nx.random_layout(flight_graph)
       )
plt.show()
```



Centrality

Centrality indices allow us to discover which nodes are the most important vertices.

The case study calculates the following four types of centrality index: **degree, PageRank, closeness, and shortest path betweenness.** We'll then augment the airports DataFrame to add a column for each of the centrality indices and visualize the results in a Mapbox map using PixieDust *display()*.

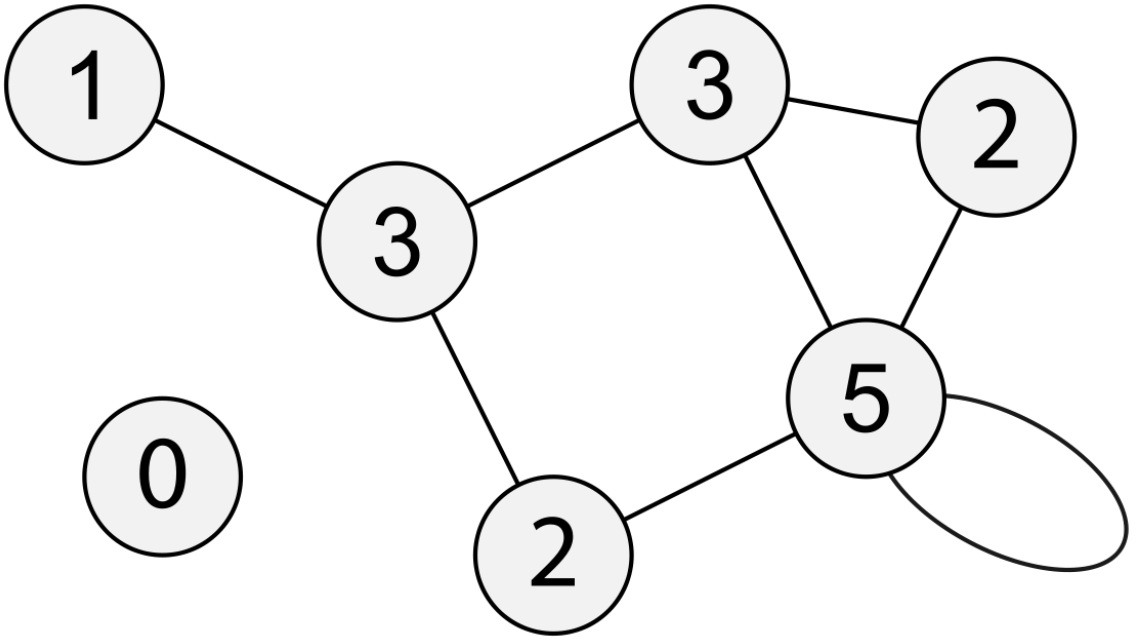
A node's **degree** is simply a count of how many social connections (i.e., edges) it has.

The vertex degrees are illustrated below for a random graph.

In [19]:

```
Image("img/US_degree.svg.png")
```

Out[19]:



PageRank is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages. According to Google:

> *PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.*

More on PageRank: https://towardsdatascience.com/graphs-and-paths-pagerank-54f180a1aa0a (https://towardsdatascience.com/graphs-and-paths-pagerank-54f180a1aa0a)
NetworkX documentation: https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html (https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html)

Closeness centrality is a way of detecting nodes that are able to spread information very efficiently through a graph.

For each node, the Closeness Centrality algorithm calculates the sum of its distances to all other nodes, based on calculating the shortest paths between all pairs of nodes. The resulting sum is then inverted to determine the closeness centrality score for that node.
https://neo4j.com/docs/graph-algorithms/current/labs-algorithms/closeness-centrality/ (https://neo4j.com/docs/graph-algorithms/current/labs-algorithms/closeness-centrality/)

Betweenness centrality is a way of detecting the amount of influence a node has over the flow of information in a graph. It is often used to find nodes that serve as a bridge from one part of a graph to another.
https://neo4j.com/blog/graph-algorithms-neo4j-betweenness-centrality/ (https://neo4j.com/blog/graph-algorithms-neo4j-betweenness-centrality/)

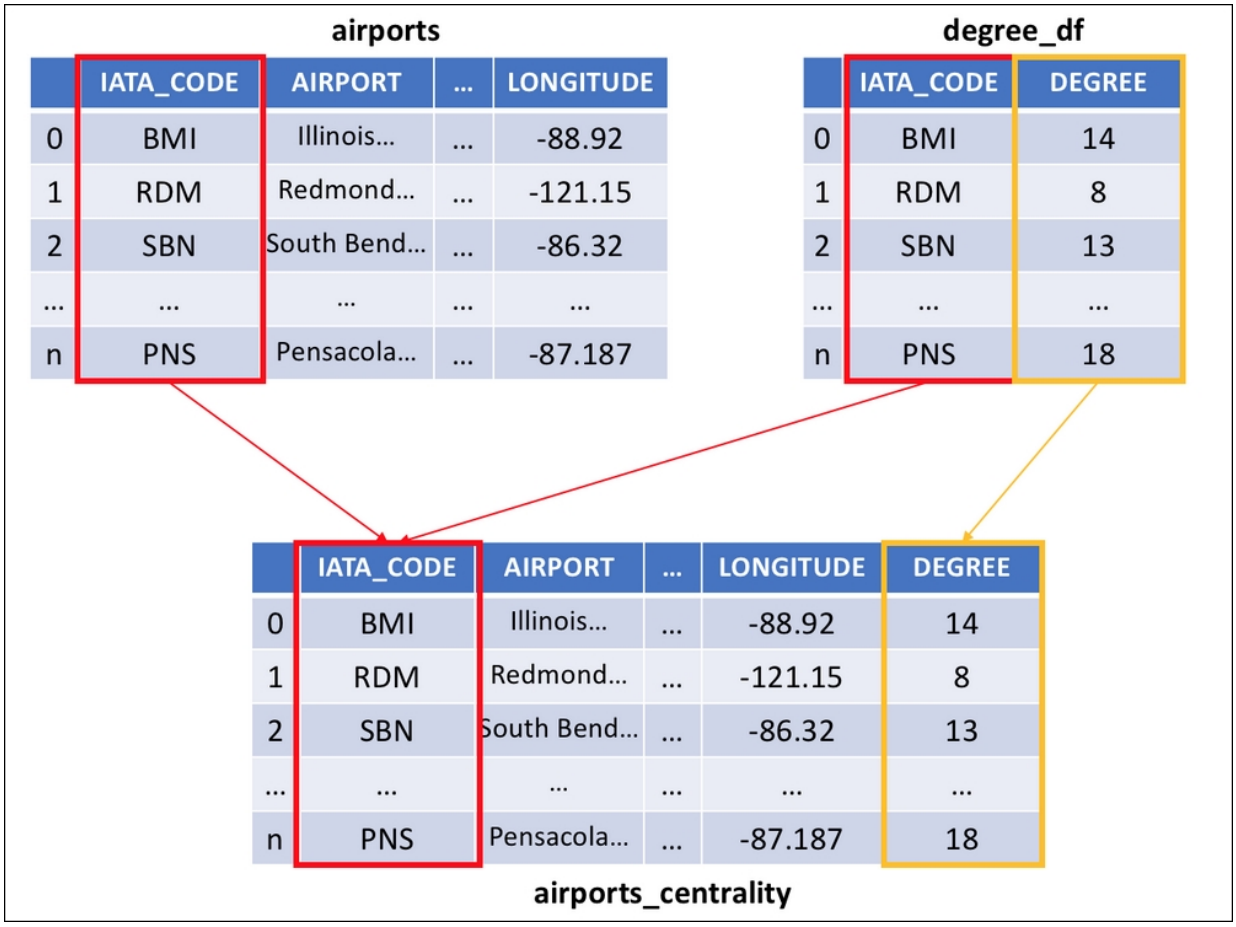
In [20]:

```
degree_df = pd.DataFrame([{"IATA_CODE":k, "DEGREE":v} for k,v in fl
```

In [21]:

```
Image("img/US_flight_merge.jpg")
```

Out[21]:



In [22]:

```
airports_centrality = pd.merge(airports, degree_df, on='IATA_CODE')
```

In [23]:

```
airports_centrality
```

Out[23]:

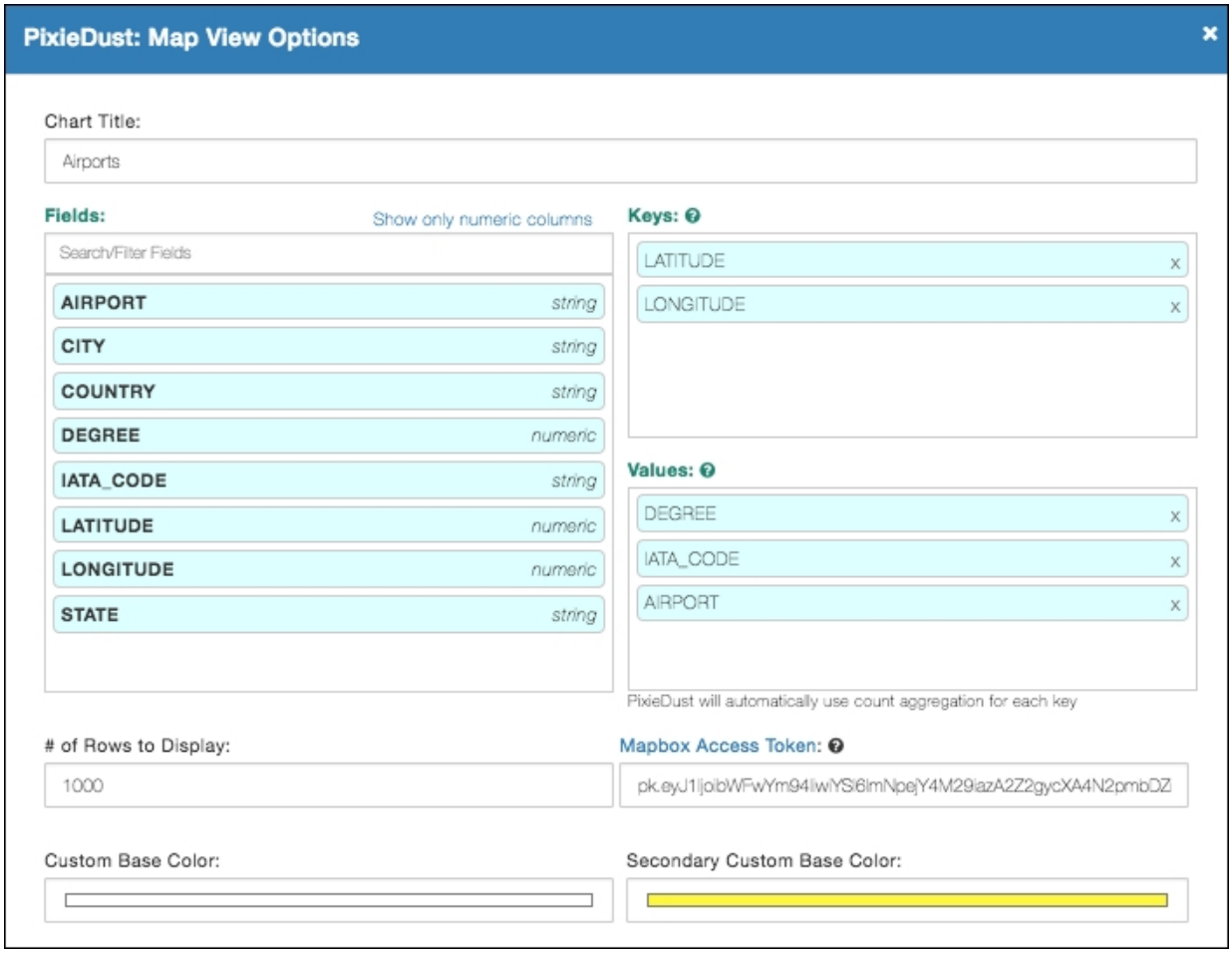
	IATA_CODE	AIRPORT	CITY	STATE	COUNTI
0	ABE	Lehigh Valley International Airport	Allentown	PA	U
1	ABI	Abilene Regional Airport	Abilene	TX	U
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	U
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	U
4	ABY	Southwest Georgia Regional Airport	Albany	GA	U
...	...	...	...	...	
317	WRG	Wrangell Airport	Wrangell	AK	U
318	WYS	Westerly State Airport	West Yellowstone	MT	U
319	XNA	Northwest Arkansas Regional Airport	Fayetteville/Springdale/Rogers	AR	U
320	YAK	Yakutat Airport	Yakutat	AK	U
321	YUM	Yuma International Airport	Yuma	AZ	U

322 rows × 8 columns

In [24]:

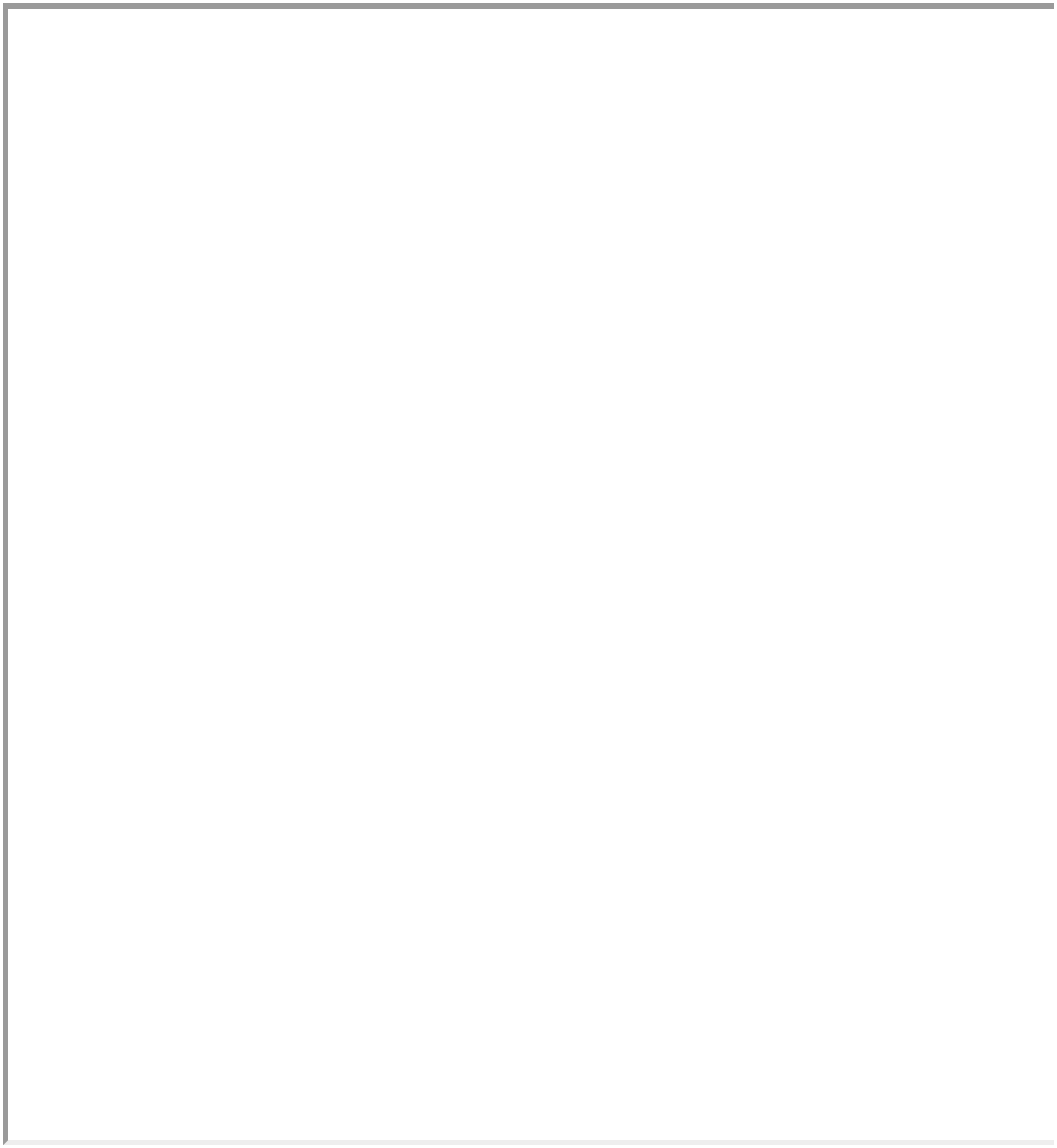
```
Image("img/US_flight_map_config.jpg")
```

Out[24]:



In [25]:

```
display(airports_centrality)
```



In [26]:

```
from six import iteritems
def compute_centrality(g, centrality_df, compute_fn, col_name, *arg
    # create a temporary DataFrame that contains the computed centr
    temp_df = pd.DataFrame(
        [{"IATA_CODE":k, col_name:v} for k,v in iteritems(compute_f
        columns=["IATA_CODE", col_name]
    )
    # make sure to remove the col_name from the centrality_df is al
    if col_name in centrality_df.columns:
        centrality_df.drop([col_name], axis=1, inplace=True)
    # merge the 2 DataFrame on the IATA_CODE column
    centrality_df = pd.merge(centrality_df, temp_df, on='IATA_CODE'
    return centrality_df
```

In [27]:

```
airports_centrality = compute_centrality(flight_graph, airports_cen
airports_centrality = compute_centrality(flight_graph, airports_cen
airports_centrality = compute_centrality(
    flight_graph, airports_centrality, nx.betweenness_centrality, "
airports_centrality
```

Out[27]:

	IATA_CODE	AIRPORT	CITY	STATE	COUNTI
0	ABE	Lehigh Valley International Airport	Allentown	PA	U
1	ABI	Abilene Regional Airport	Abilene	TX	U
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	U
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	U
4	ABY	Southwest Georgia Regional	Albany	GA	U

		Airport			
...	...	...	...	...	
317	WRG	Wrangell Airport	Wrangell	AK	U
318	WYS	Westerly State Airport	West Yellowstone	MT	U
319	XNA	Northwest Arkansas Regional Airport	Fayetteville/Springdale/Rogers	AR	U
320	YAK	Yakutat Airport	Yakutat	AK	U
321	YUM	Yuma International Airport	Yuma	AZ	U

322 rows × 11 columns

In [28]:

```
for col_name in ["DEGREE", "PAGE_RANK", "CLOSENESS", "BETWEENNESS"]
    print("{} : {}".format(
        col_name,
        airports_centrality.nlargest(10, col_name)["IATA_CODE"].val
    )
```

```
DEGREE : ['ATL' 'ORD' 'DFW' 'DEN' 'MSP' 'IAH' 'DTW' 'S
LC' 'EWR' 'LAX']
PAGE_RANK : ['ATL' 'ORD' 'DFW' 'DEN' 'MSP' 'IAH' 'DTW'
'SLC' 'SFO' 'LAX']
CLOSENESS : ['ATL' 'ORD' 'DFW' 'DEN' 'MSP' 'IAH' 'DTW'
'SLC' 'EWR' 'LAX']
BETWEENNESS : ['ATL' 'DFW' 'ORD' 'DEN' 'MSP' 'SLC' 'DT
W' 'ANC' 'IAH' 'SFO']
```

visualize_neighbors()

visualizes all the neighbors of a given node.

In this method, we create a subgraph centered around the parent node by adding an edge from itself to all its neighbors.

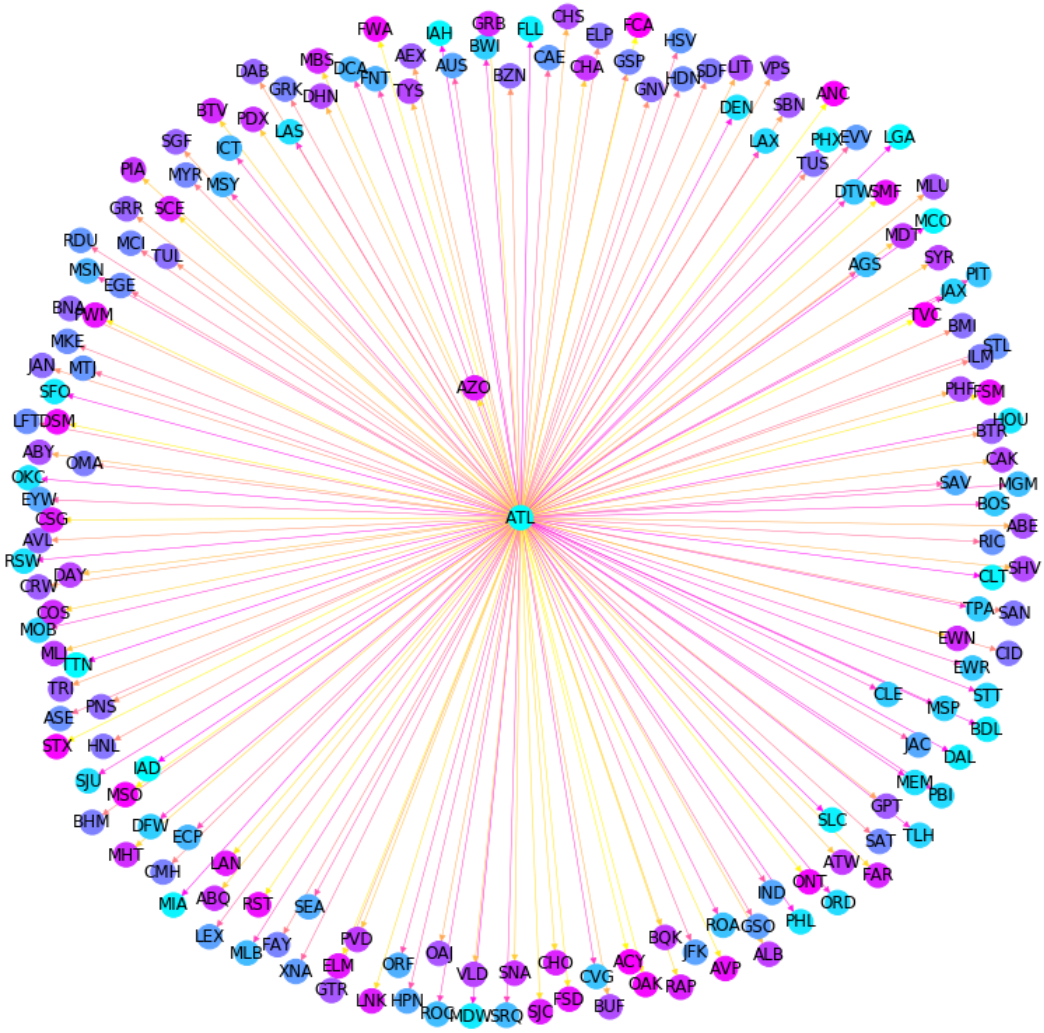
We use the NetworkX ***neighbors()*** method to get all the neighbors of a specific node.

In [29]:

```
import matplotlib.cm as cm
def visualize_neighbors(parent_node):
    fig = plt.figure(figsize = (12,12))
    # Create a subgraph and add an edge from the parents node to al
    graph = nx.DiGraph()
    for neighbor in flight_graph.neighbors(parent_node):
        graph.add_edge(parent_node, neighbor)
    # draw the subgraph
    nx.draw(graph, arrows=True,
            with_labels=True,
            width = 0.5,
            style="dotted",
            node_color=range(len(graph)),
            cmap=cm.get_cmap(name="cool"),
            edge_color=range(len(graph.edges)),
            edge_cmap=cm.get_cmap(name="spring"),
           )
    plt.show()
```

In [30]:

```
visualize_neighbors("ATL")
```



In [ ]:

```

```

In [ ]:

```

```