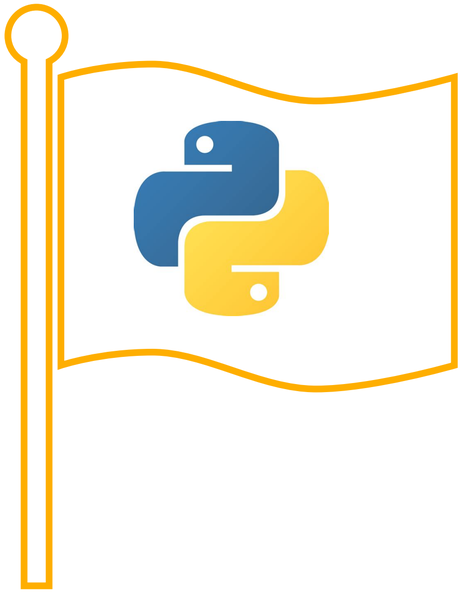
PythOn-ward to Victory



Michael Colaresi

Why Python?

- Python is a general purpose coding language
- Huge user community
- Pretty easy learning curve
  - That will take you far…
    - Application development…etc
- A bit slow
  - But there are fixes for that
    - Like knowing how your computer works!

Python 2 vs 3

- **A note about version numbers**
- For most software, there is a 3 digit code to understand versions (Semantic versioning)
    - Take the example. 3.5.7
- The first number to the left is the **major** version number (3.X.X)
    - Major version numbers do not increment very often
    - It is not expected that code that runs on one major version number of software will work with another major version.
    - Python 2 was the last major version of Python then
    - Python 3 is the current major version of Python…. It broke backwards compatibility
        - Python 3 is it now. You should really not be using Python 2
- The second number is the **minor** version number (X.5.X)
    - X.5 should generally work with X.4 code, but may have minor features added
    - Python 3.7 is the 7th minor release after the origin Python 3.0/Python 3
- The third number is the **patch** version number (X.X.7)
    - Very small changes are suggested between X.X.6 and X.X.7
    - Python 3.7.3 is the 3rd patch to Python 3.7.

Python 2 vs 3

- **A note about version numbers**
- For most software, there is a 3 digit code to understand versions (Semantic versioning)
  - Take the example. 3.5.7
- The first number to the left is the **major** version number (3.X.X)
  - Major version numbers do not increment very often
  - It is not expected that code that runs on one major version number of software will work with another major version.
  - Python 2 was the last major version of Python then
  - Python 3 is the current major version of Python…. It broke backwards compatibility
    - Python 3 is it now. You should really not be using Python 2
- The second number is the **minor** version number (X.5.X)
  - X.5 should generally work with X.4 code, but may have minor features added
  - Python 3.7 is the 7th minor release after the origin Python 3.0/Python 3
- The third number is the **patch** version number (X.X.7)
  - Very small changes are suggested between X.X.6 and X.X.7
  - Python 3.7.3 is the 3rd patch to Python 3.7.

There are really two ways to see code:
Private/behind the scenes
And Public

In private, code is being changed all the time by teams (think commits and branches on github)

In public, there are fewer, but still potentially numerous releases.

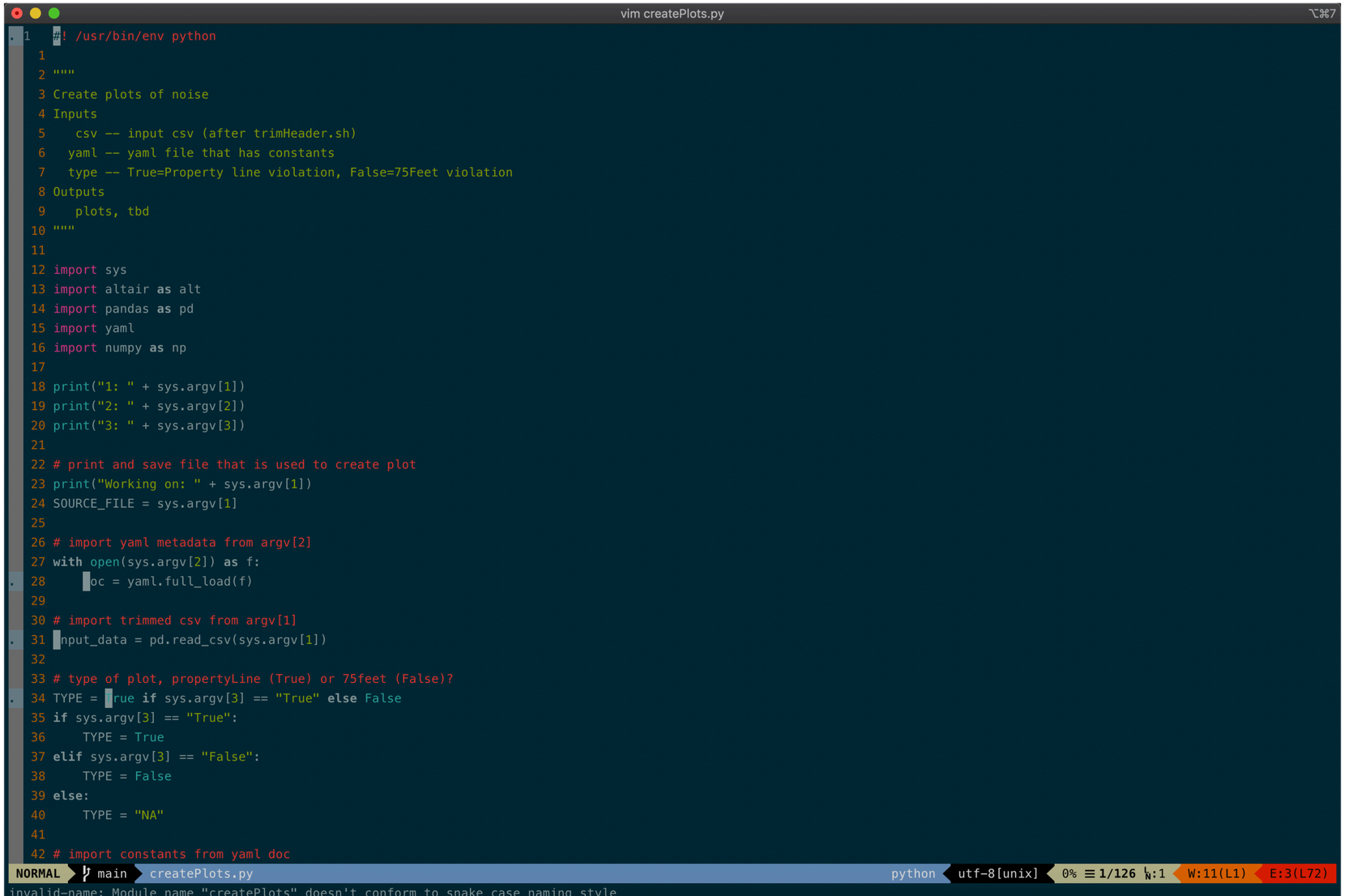
Semantic versioning is really about the public releases.

Speaking of counting

- Python starts indexing at 0
  - So when we index things…
    - The “first” item is at slot 0
    - The “second” item is at slot 1
    - Why waste a perfectly good number?

Lets breakdown a simple Python script

Bring the Noise! (example)



vim createPlots.py ⌥⌘7

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

NORMAL main createPlots.py python utf-8[unix] 0% ☰ 1/126 ㏑:1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

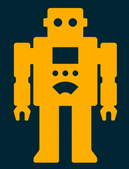
vim createPlots.py ⌥⌘7

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHe
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

1

#! /usr/bin/env python

The familiar shebang



NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader
  yaml -- yaml file that has constants
  type -- True=Property line violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""
```

A “docstring”: multiline comment

What is this script doing? Inputs outputs

NORMAL main createPlots.py python utf-8[unix] 0% 1/126 :1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.
  yaml -- yaml file that has constants
  type -- True=Property line violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
NORMAL  main  createPlots.py                    python  utf-8[unix]  0% ≡ 1/126 ㏑:1  W:11(L1)  E:3(L72)
nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

A “docstring”: multiline comment

```
"""
Create plots of noise
Inputs
    csv -- input csv (after trimHeader.sh)
   yaml -- yaml file that has constants
   type -- True=Property line violation, False=75Feet violation
Outputs
    plots, tbd
"""
```

Three quotes for a block comment.

Docstrings like this should go toward the top of scripts
At the top of functions
And at the top of class definitions

vim createPlots.py ⌥⌘7

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np
```

Import “modules”: libraries of other peoples code

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

vim createPlots.py ⌥⌘7

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    doc = yaml.full_load(f)

# import trimmed csv from argv[1]
input_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = True if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np
```

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

Invalid-name: Module name "createPlots" doesn't conform to snake_case naming style

Import “modules”: libraries of other peoples code

Two types of syntax:

1. Just `import ….` — then use name “sys” to access functions and classes

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    doc = yaml.full_load(f)

# import trimmed csv from argv[1]
input_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = True if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np
```

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

Invalid-name: Module name "createPlots" doesn't conform to snake_case naming style

Import “modules”: libraries of other peoples code

Two types of syntax:

1. Just `import ….` — then use name “sys” to access functions and classes

2. Import something as abbrv — so then you can just use abbrv for short

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np
```

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

Invalid-name: Module name "createPlots" doesn't conform to snake_case naming style

Import “modules”: libraries of other peoples code

Two types of syntax:

1. Just `import ….` — then use name “sys” to access functions and classes

2. Import something as abbrv — so then you can just use abbrv for short

If you do not want to get your lunch money stolen at Python recess you will:

- import pandas as pd,
- numpy as np, and follow other conventions like this so code is inter-operable.

vim createPlots.py ⌥⌘7

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

Some REAL Python syntax!

```
17
18 print("1: " + sys.argv[1])
19 print("2: " + sys.argv[2])
20 print("3: " + sys.argv[3])
21
```

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
17
18 print("1: " + sys.argv[1])
19 print("2: " + sys.argv[2])
20 print("3: " + sys.argv[3])
21
```

Some REAL Python syntax!

print() is a function

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

Invalid-name: Module name "createPlots" doesn't conform to snake_case naming style

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
17
18 print("1: " + sys.argv[1])
19 print("2: " + sys.argv[2])
20 print("3: " + sys.argv[3])
21
```

Some REAL Python syntax!

Quotes are defining strings
You can use double or single

```
NORMAL  main  createPlots.py                    python  utf-8[unix]  0% ≡ 1/126 ㏑:1  W:11(L1)  E:3(L72)
Invalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

Some REAL Python syntax!

```
17
18 print("1: " + sys.argv[1])
19 print("2: " + sys.argv[2])
20 print("3: " + sys.argv[3])
21
```

sys.argv

sys : we imported that!

```
NORMAL  main  createPlots.py                    python  utf-8[unix]  0% ≡ 1/126 ㏑:1  W:11(L1)  E:3(L72)
nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
17
18 print("1: " + sys.argv[1])
19 print("2: " + sys.argv[2])
20 print("3: " + sys.argv[3])
21
```

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

Some REAL Python syntax!

sys.argv

sys : we imported that!

argv is a function within the sys module

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
17
18 print("1: " + sys.argv[1])
19 print("2: " + sys.argv[2])
20 print("3: " + sys.argv[3])
21
```

NORMAL main createPlots.py python utf-8[unix] 0% 1/126 :1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

Some REAL Python syntax!

sys.argv

sys : we imported that!

argv is a function within the sys module

It grabs arguments from the shell
when the script is run

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
17
18 print("1: " + sys.argv[1])
19 print("2: " + sys.argv[2])
20 print("3: " + sys.argv[3])
21
```

Some REAL Python syntax!

sys.argv

sys : we imported that!

argv is a function within the sys module

A module is just a directory/script with functions and class definitions!

> python createPlots.py Arg1, Arg2, …

```
NORMAL  main  createPlots.py                    python  utf-8[unix]  0%  1/126  :1  W:11(L1)  E:3(L72)
nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    doc = yaml.full_load(f)

# import trimmed csv from argv[1]
input_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = True if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

Some REAL Python syntax!

```
17
18 print("1: " + sys.argv[1])
19 print("2: " + sys.argv[2])
20 print("3: " + sys.argv[3])
21
```

sys.argv

Is a list

[Arg0, Arg1, Arg2, Arg3, …]
sys.argv[1] is Arg1

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

Invalid-name: Module name "createPlots" doesn't conform to snake_case naming style

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
17
18 print("1: " + sys.argv[1])
19 print("2: " + sys.argv[2])
20 print("3: " + sys.argv[3])
21
```

NORMAL main createPlots.py python utf-8[unix] 0% 1/126 :1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

Some REAL Python syntax!

sys.argv

Is a list

[Arg0, Arg1, Arg2, Arg3, …]
sys.argv[1] is Arg1

A = [“apple”, “linux”, “WSL”]
print(A[1])
“linux”
print(A[2])
“WSL”

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = rue if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

Some REAL Python syntax!

```
17
18 print("1: " + sys.argv[1])
19 print("2: " + sys.argv[2])
20 print("3: " + sys.argv[3])
21
```

\+ is not addition here
Concatenating 2 strings!

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

Invalid-name: Module name "createPlots" doesn't conform to snake_case naming style

vim createPlots.py ⌥⌘7

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    doc = yaml.full_load(f)

# import trimmed csv from argv[1]
input_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = True if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
NORMAL  main  createPlots.py                    python  utf-8[unix]  0%  1/126  :1  W:11(L1)  E:3(L72)
nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

Some REAL Python syntax!

```
17
18 print("1: " + sys.argv[1])
19 print("2: " + sys.argv[2])
20 print("3: " + sys.argv[3])
21
```

\+ is not addition here
Concatenating 2 strings!

“test this” + “ out”
“test this out”

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    doc = yaml.full_load(f)

# import trimmed csv from argv[1]
input_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = True if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

Some REAL Python syntax!

```
22 # print and save file that is used to create plot
23 print("Working on: " + sys.argv[1])
24 SOURCE_FILE = sys.argv[1]
```

Assigning the value in sys.argv[1] to the variable SOURCE_FILE

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

Invalid-name: Module name "createPlots" doesn't conform to snake_case naming style

vim createPlots.py

```
#! /usr/bin/env python
"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    doc = yaml.full_load(f)

# import trimmed csv from argv[1]
input_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
TYPE = True if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
NORMAL  main  createPlots.py                    python  utf-8[unix]  0%  1/126  :1  W:11(L1)  E:3(L72)
Invalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

Some REAL Python syntax!

```
# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]
```

Assigning the value in sys.argv[1] to the variable SOURCE_FILE

Python is dynamically typed!

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or
TYPE = True if sys.argv[3] == "True" e
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

Some REAL Python syntax!

Open a file connection (we are going to read a yaml file)

```
26 # import yaml metadata from argv[2]
27 with open(sys.argv[2]) as f:
28     oc = yaml.full_load(f)
29
```

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or
TYPE = True if sys.argv[3] == "True" e
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

Some REAL Python syntax!

Open a file connection (we are going to read a yaml file)

```
26 # import yaml metadata from argv[2]
27 with open(sys.argv[2]) as f:
28     oc = yaml.full_load(f)
29
```

`with` specifies the scope of the next commands

```
NORMAL  main  createPlots.py                python  utf-8[unix]  0% ≡ 1/126 ㏑:1  W:11(L1)  E:3(L72)
nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or
TYPE = rue if sys.argv[3] == "True" e
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

NORMAL main createPlots.py python utf-8[unix] 0% 1/126 :1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

Some REAL Python syntax!

Open a file connection (we are going to read a yaml file)

```
26 # import yaml metadata from argv[2]
27 with open(sys.argv[2]) as f:
28     oc = yaml.full_load(f)
29
```

`with` specifies the scope of the next commands

open is a function that takes a path to file/memory

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or
TYPE = True if sys.argv[3] == "True" e
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

Some REAL Python syntax!

Open a file connection (we are going to read a yaml file)

```
26 # import yaml metadata from argv[2]
27 with open(sys.argv[2]) as f:
28     oc = yaml.full_load(f)
29
```

`with` specifies the scope of the next commands

open is a function that takes a path to file/memory

f is the name of the file that is being connected to

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or
TYPE = rue if sys.argv[3] == "True" e
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

NORMAL main createPlots.py python utf-8[unix] 0% 1/126 :1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

Some REAL Python syntax!

Once indentation ends…. f is closed!
This is awesome… like some shutting
Off the oven for you so you do not
burn down the house.
Or hanging up the phone, less
dramatically

Open a file connection (we are going to read a yaml file)

```
26 # import yaml metadata from argv[2]
27 with open(sys.argv[2]) as f:
28     oc = yaml.full_load(f)
29
```

`with` specifies the scope of the next commands

open is a function that takes a path to file/memory

f is the name of the file that is being connected to

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    doc = yaml.full_load(f)

# import trimmed csv from argv[1]
input_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or
TYPE = True if sys.argv[3] == "True" e
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

Some REAL Python syntax!

Open a file connection (we are going to read a yaml file)

```
26 # import yaml metadata from argv[2]
27 with open(sys.argv[2]) as f:
28     doc = yaml.full_load(f)
29
```

Creating a variable `doc`, using the yaml.full_load() function

```
NORMAL  main  createPlots.py                python  utf-8[unix]  0%  1/126  :1  W:11(L1)  E:3(L72)
invalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    doc = yaml.full_load(f)

# import trimmed csv from argv[1]
input_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or
TYPE = True if sys.argv[3] == "True" e
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

Some REAL Python syntax!

Open a file connection (we are going to read a yaml file)

```
26 # import yaml metadata from argv[2]
27 with open(sys.argv[2]) as f:
28     doc = yaml.full_load(f)
29
```

Creating a variable `doc`, using the yaml.full_load() function
Reading the yaml file given as an argument

```
NORMAL  main  createPlots.py                    python  utf-8[unix]  0%  1/126  :1  W:11(L1)  E:3(L72)
nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True)
TYPE = True if sys.argv[3] == "True"
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

Some REAL Python syntax!

Use pandas (as pd) to read the csv data given in sys.argv[1]

```
30 # import trimmed csv from argv[1]
31 nput_data = pd.read_csv(sys.argv[1])
32
```

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
    csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
    plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    doc = yaml.full_load(f)

# import trimmed csv from argv[1]
input_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True)
TYPE = True if sys.argv[3] == "True"
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

Some REAL Python syntax!

Use pandas (as pd) to read the csv data given in sys.argv[1]

```
30 # import trimmed csv from argv[1]
31 input_data = pd.read_csv(sys.argv[1])
32
```

Assign that to variable input_data

```
NORMAL  main  createPlots.py                    python  utf-8[unix]  0% ≡ 1/126 ㏑:1  W:11(L1)  E:3(L72)
nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (Fa
TYPE = True if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

Some REAL Python syntax!

Ternary operator: This if CONDITION TRUE else That

```
# type of plot, propertyLine (True) or 75feet (False)?
TYPE = True if sys.argv[3] == "True" else False
```

Assigning the output of a ternary operator to a variable

```
NORMAL  main  createPlots.py                    python  utf-8[unix]  0% ≡ 1/126 ㏑:1  W:11(L1)  E:3(L72)
nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (Fa
TYPE = True if sys.argv[3] == "True" else False
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

Some REAL Python syntax!

Ternary operator: This if CONDITION TRUE else That

```
33 # type of plot, propertyLine (True) or 75feet (False)?
34 TYPE = True if sys.argv[3] == "True" else False
```

Assigning the output of a ternary operator to a variable

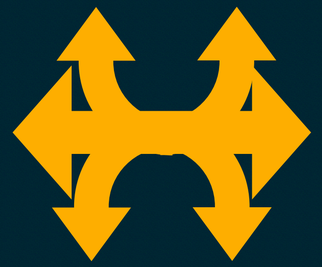
Note the “quotes” around “True”; this is because sys.argv passes strings

```
NORMAL  main  createPlots.py                    python  utf-8[unix]  0% ≡ 1/126 ㏑:1  W:11(L1)  E:3(L72)
nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from ar
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv
nput_data = pd.read_csv(sys.a

# type of plot, propertyLine (
TYPE = True if sys.argv[3] ==
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"
```



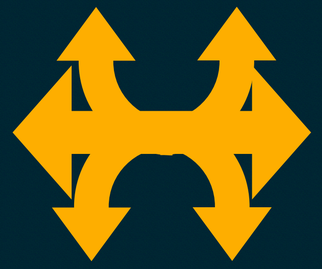
Some REAL Python syntax!

Conditional flow, if, elif, else

```
if CONDITION:
    What to do if CONDITION is True
elif ANOTHER_CONDITION:
    What to do if  ANOTHER_CONDITION is True:
else:
    What to do if both are False
```

NORMAL main createPlots.py python utf-8[unix] 0% ≡ 1/126 ㏑:1 W:11(L1) E:3(L72)

nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style



Some REAL Python syntax!

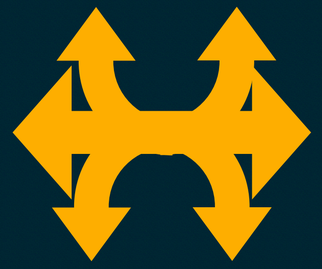
Notice the syntax and whitespace

Conditional flow, if, elif, else

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from ar
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv
nput_data = pd.read_csv(sys.a

# type of plot, propertyLine (
TYPE = True if sys.argv[3] ==
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"

# import constants from yaml doc
```

```
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"
```


nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style



Some REAL Python syntax!

Notice the syntax and whitespace

indentation is a unique python requirement

Conditional flow, if, elif, else

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
    csv -- input csv (after trimHeader.sh)
   yaml -- yaml file that has constants
   type -- True=Property line violation, False=75Feet violation
Outputs
    plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from ar
with open(sys.argv[2]) as f:
    oc = yaml.full_load(f)

# import trimmed csv from argv
nput_data = pd.read_csv(sys.a

# type of plot, propertyLine (
TYPE = True if sys.argv[3] ==
```

```
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"
```

```
# import constants from yaml doc
```

```
if sys.argv[3] == "True":
    TYPE = True
elif sys.argv[3] == "False":
    TYPE = False
else:
    TYPE = "NA"
```

```
NORMAL  main  createPlots.py                    python  utf-8[unix]  0%  1/126  :1  W:11(L1)  E:3(L72)
nvalid-name: Module name "createPlots" doesn't conform to snake_case naming style
```

Let’s pause and consolidate our gains

comments, long (“””) and short (#)

“import”/“import as” other peoples code into memory/imagination

print is a function to echo to std out

Pipe argument from the command line into python scripts using sys.argv

Lists, like yaml seq (defined with “[“ “]”), access elements, eg aList[3]

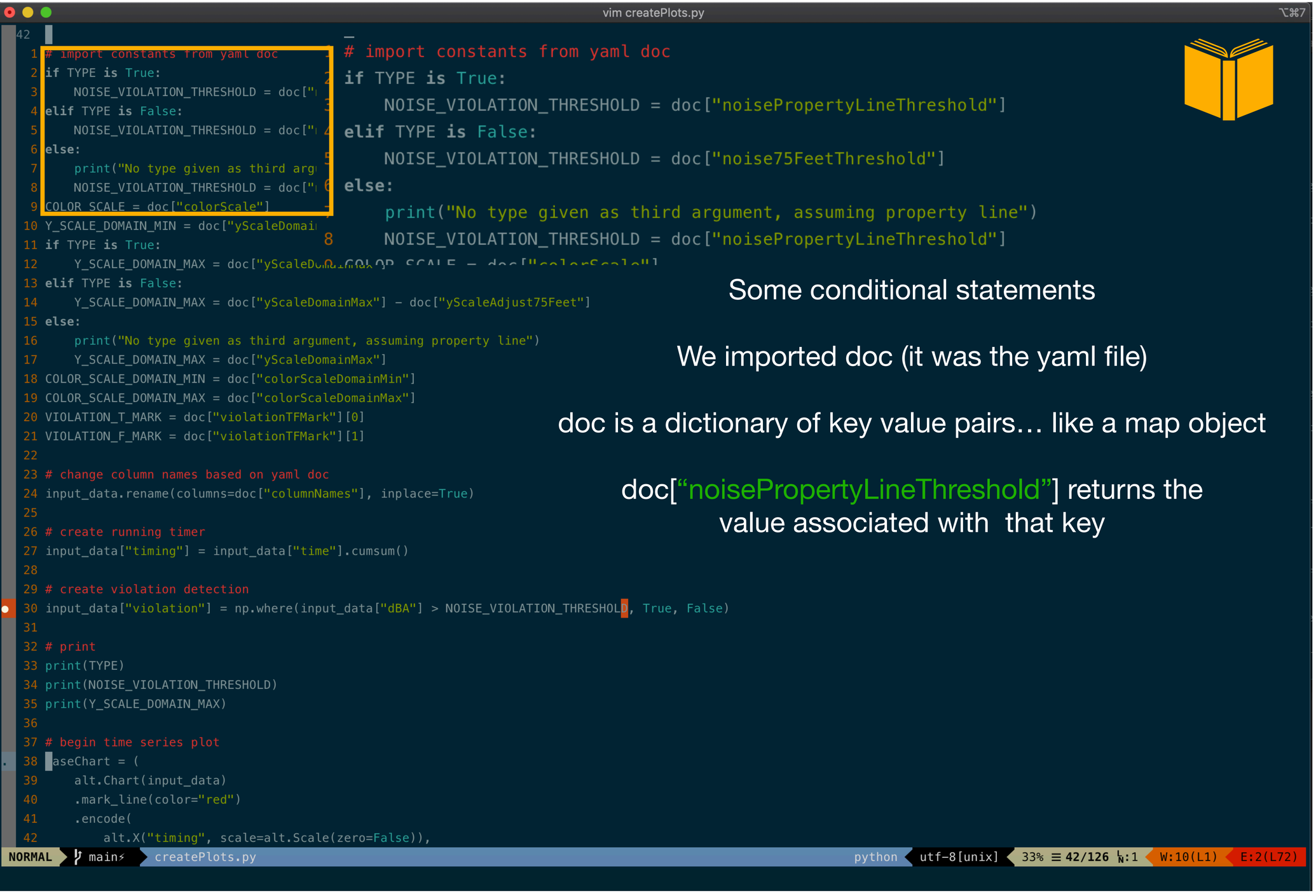
Concatenate strings with “+”

Assign values to objects, “=“ (like pointer from a name to a value)

Call up files from computer storage (open())

Read the information in files “yaml.fullLoad()” and “pd.read_csv()”

Control flow (ternary __ if COND else ____) and if, elif, elif, … ,else…


vim createPlots.py
# import constants from yaml doc
if TYPE is True:
NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
print("No type given as third argument, assuming property line")
NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
print("No type given as third argument, assuming property line")
Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]
# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)
# create running timer
input_data["timing"] = input_data["time"].cumsum()
create violation detection
input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)
# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)
# begin time series plot
baseChart = (
alt.Chart(input_data)
.mark_line(color="red")
.encode(
alt.X("timing", scale=alt.Scale(zero=False)),
NORMAL main createPlots.py python utf-8[unix] 33% 42/126 :1 W:10(L1) E:2(L72)
Some conditional statements
We imported doc (it was the yaml file)
doc is a dictionary of key value pairs… like a map object
doc[“noisePropertyLineThreshold”] returns the value associated with that key

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]

# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time"].cumsum()

# create violation detection
input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
baseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

Dictionary name

Some conditional statements

We imported doc (it was the yaml file)

doc is a dictionary of key value pairs… like a map object

doc[“noisePropertyLineThreshold”] returns the value associated with that key

NORMAL main createPlots.py python utf-8[unix] 33% 42/126 :1 W:10(L1) E:2(L72)

vim createPlots.py

```python
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]

# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time"].cumsum()

# create violation detection
input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
baseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

Key

Some conditional statements

We imported doc (it was the yaml file)

doc is a dictionary of key value pairs… like a map object

doc[“noisePropertyLineThreshold”] returns the value associated with that key

NORMAL main createPlots.py python utf-8[unix] 33% 42/126 :1 W:10(L1) E:2(L72)

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]

# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time"].cumsum()

# create violation detection
input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
baseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

Key

"noisePropertyLineThreshold"

Points to…

a value

that was in yaml file

Some conditional statements

We imported doc (it was the yaml file)

doc is a dictionary of key value pairs… like a map object

doc[“noisePropertyLineThreshold”] returns the value associated with that key

NORMAL main⚡ createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

vim createPlots.py

```
42
1  # import constants from yaml doc
2  if TYPE is True:
3      NOISE_VIOLATION_THRESHOLD = doc["
4  elif TYPE is False:
5      NOISE_VIOLATION_THRESHOLD = doc["
6  else:
7      print("No type given as third arg
8      NOISE_VIOLATION_THRESHOLD = doc["
9  COLOR_SCALE = doc["colorScale"]
10 Y_SCALE_DOMAIN_MIN = doc["yScaleDomai
11 if TYPE is True:
12     Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
13 elif TYPE is False:
14     Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
15 else:
16     print("No type given as third argument, assuming property line")
17     Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
18 COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
19 COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
20 VIOLATION_T_MARK = doc["violationTFMark"][0]
21 VIOLATION_F_MARK = doc["violationTFMark"][1]
22
23 # change column names based on yaml doc
24 input_data.rename(columns=doc["columnNames"], inplace=True)
25
26 # create running timer
27 input_data["timing"] = input_data["time"].cumsum()
28
29 # create violation detection
30 input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)
31
32 # print
33 print(TYPE)
34 print(NOISE_VIOLATION_THRESHOLD)
35 print(Y_SCALE_DOMAIN_MAX)
36
37 # begin time series plot
38 baseChart = (
39     alt.Chart(input_data)
40     .mark_line(color="red")
41     .encode(
42         alt.X("timing", scale=alt.Scale(zero=False)),
```

```
1  # import constants from yaml doc
2  if TYPE is True:
3      NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
4  elif TYPE is False:
5      NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
6  else:
7      print("No type given as third argument, assuming property line")
8      NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
```

Key

Points to…

a value

that was in yaml file

Some conditional statements

We imported doc (it was the yaml file)

doc is a dictionary of key value pairs… like a map object

doc[“noisePropertyLineThreshold”] returns the value associated with that key

In python we create dictionary, with
my_dict = {“key1”: val1, “key2”: val2, etc }

NORMAL main createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

vim createPlots.py

```
42
1  # import constants from yaml doc
2  if TYPE is True:
3      NOISE_VIOLATION_THRESHOLD = doc["
4  elif TYPE is False:
5      NOISE_VIOLATION_THRESHOLD = doc["
6  else:
7      print("No type given as third arg
8      NOISE_VIOLATION_THRESHOLD = doc["
9  COLOR_SCALE = doc["colorScale"]
10 Y_SCALE_DOMAIN_MIN = doc["yScaleDomai
11 if TYPE is True:
12     Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
13 elif TYPE is False:
14     Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
15 else:
16     print("No type given as third argument, assuming property line")
17     Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
18 COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
19 COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
20 VIOLATION_T_MARK = doc["violationTFMark"][0]
21 VIOLATION_F_MARK = doc["violationTFMark"][1]
22
23 # change column names based on yaml doc
24 input_data.rename(columns=doc["columnNames"], inplace=True)
25
26 # create running timer
27 input_data["timing"] = input_data["time"].cumsum()
28
29 # create violation detection
30 input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)
31
32 # print
33 print(TYPE)
34 print(NOISE_VIOLATION_THRESHOLD)
35 print(Y_SCALE_DOMAIN_MAX)
36
37 # begin time series plot
38 baseChart = (
39     alt.Chart(input_data)
40     .mark_line(color="red")
41     .encode(
42         alt.X("timing", scale=alt.Scale(zero=False)),
```

```
1  # import constants from yaml doc
2  if TYPE is True:
3      NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
4  elif TYPE is False:
5      NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
6  else:
7      print("No type given as third argument, assuming property line")
8      NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
```

Key

Points to…

a value

that was in yaml file

Some conditional statements

We imported doc (it was the yaml file)

doc is a dictionary of key value pairs… like a map object

doc[“noisePropertyLineThreshold”] returns the value associated with that key

In python we create dictionary, with
my_dict = {“key1”: val1, “key2”: val2, etc }

In this case we read doc in as a dictionary from a yaml map

NORMAL main createPlots.py python utf-8[unix] 33% 42/126 :1 W:10(L1) E:2(L72)

vim createPlots.py

```python
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomai
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleD
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleD
else:
    print("No type given as third arg
    Y_SCALE_DOMAIN_MAX = doc["yScaleD
COLOR_SCALE_DOMAIN_MIN = doc["colorSc
COLOR_SCALE_DOMAIN_MAX = doc["colorSc
VIOLATION_T_MARK = doc["violationTFMa
VIOLATION_F_MARK = doc["violationTFMa

# change column names based on yaml d
input_data.rename(columns=doc["column

# create running timer
input_data["timing"] = input_data["ti

# create violation detection
input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
aseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

```python
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]
```

Key → a value that is a list

Points to…

The value of a dictionary can be just about anything….

NORMAL main⚡ createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomai
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleD
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleD
else:
    print("No type given as third arg
    Y_SCALE_DOMAIN_MAX = doc["yScaleD
COLOR_SCALE_DOMAIN_MIN = doc["colorSc
COLOR_SCALE_DOMAIN_MAX = doc["colorSc
VIOLATION_T_MARK = doc["violationTFMa
VIOLATION_F_MARK = doc["violationTFMa

# change column names based on yaml d
input_data.rename(columns=doc["column

# create running timer
input_data["timing"] = input_data["ti

# create violation detection
input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
aseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

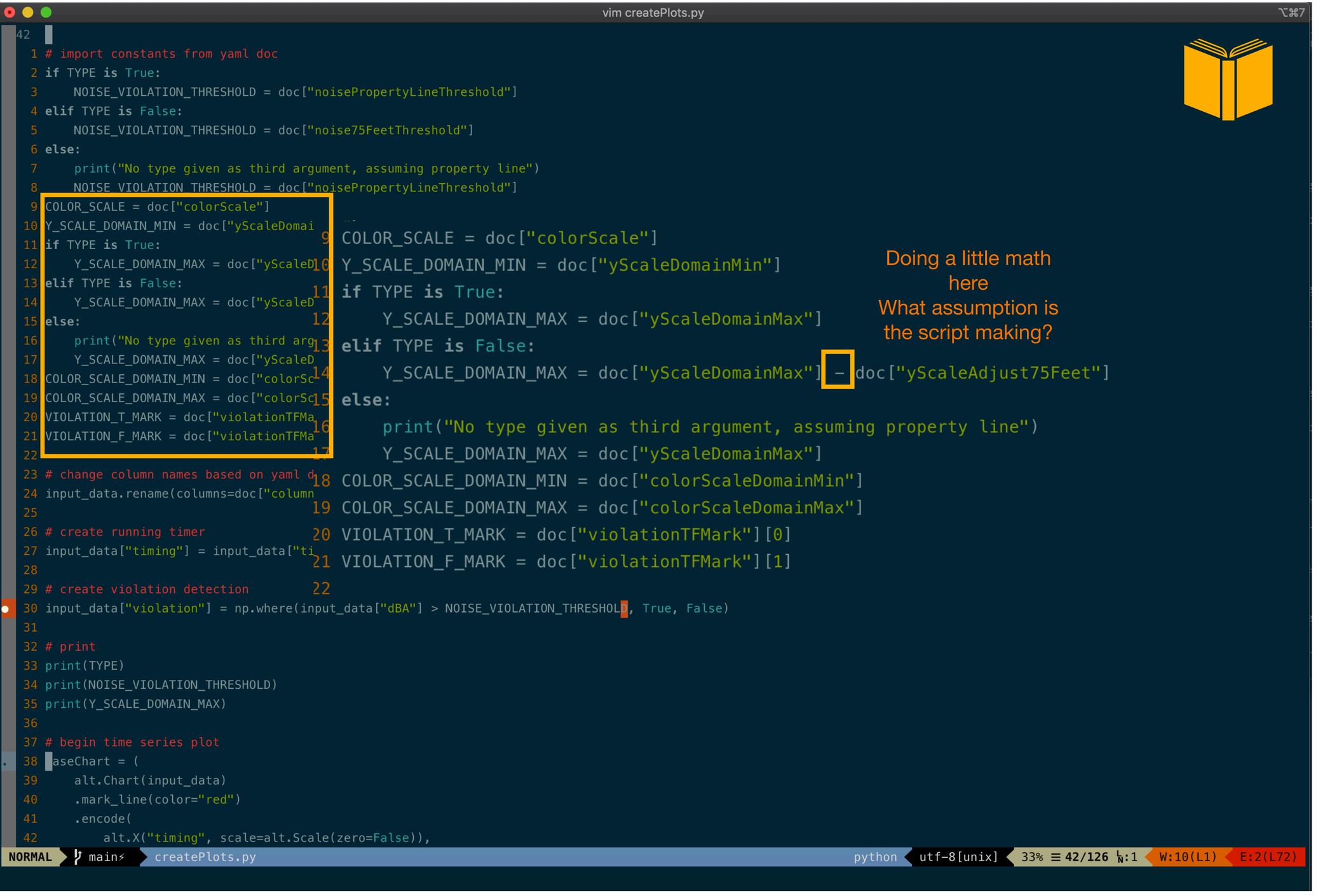
```
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]
```

Key → Points to… a value that is a list

And we are grabbing the second element

The value of a dictionary can be just about anything….

NORMAL main⚡ createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)


vim createPlots.py
# import constants from yaml doc
if TYPE is True:
NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
print("No type given as third argument, assuming property line")
NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
print("No type given as third argument, assuming property line")
Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]
Doing a little math here
What assumption is the script making?
change column names based on yaml d
input_data.rename(columns=doc["column
create running timer
input_data["timing"] = input_data["ti
create violation detection
input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)
# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)
# begin time series plot
aseChart = (
alt.Chart(input_data)
.mark_line(color="red")
.encode(
alt.X("timing", scale=alt.Scale(zero=False)),
NORMAL main createPlots.py python utf-8[unix] 33% 42/126 :1 W:10(L1) E:2(L72)

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]

# change column names based on yaml doc
input_data.rename(columns=doc["columnNa

# create running timer
input_data["timing"] = input_data["time

# create violation detection
input_data["violation"] = np.where(inpu

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
aseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

Is a pd.DataFrame
Like a cool excel spreadsheet (can think about rows and columns, but it is more general than that)

```
# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time"].cumsum()
```

NORMAL main⚡ createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]

# change column names based on yaml doc
input_data.rename(columns=doc["columnNa

# create running timer
input_data["timing"] = input_data["time

# create violation detection
input_data["violation"] = np.where(inpu

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
aseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

```
# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time"].cumsum()
```

Is a pd.DataFrame
Like a cool excel spreadsheet (can think about rows and columns, but it is more general than that)

We say that this is the object's type

NORMAL main createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]

# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time"].cumsum()

# create violation detection
input_data["violation"] = np.where(input_data["violation"] == True, VIOLATION_T_MARK, VIOLATION_F_MARK)

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
baseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

Objects can have methods associated with them… methods are like functions but they take the object as the first argument.

```
NORMAL  main⚡  createPlots.py                    python  utf-8[unix]  33% ≡ 42/126 ㏑:1  W:10(L1)  E:2(L72)
```

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]

# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time"].cumsum()

# create violation detection
input_data["violation"] = np.where(input_data["violation"] == VIOLATION_T_MARK, True, False)

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
baseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

Objects can have methods associated with them… methods are like functions but they take the object as the first argument.

The syntax is obj.method(arguments here)

NORMAL main createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]


# change column names based on yaml doc
input_data.rename(columns=doc["columnNa

# create running timer
input_data["timing"] = input_data["time

# create violation detection
input_data["violation"] = np.where(inpu

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
aseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

Objects can have methods associated with them… methods are like functions but they take the object as the first argument.

```
# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time"].cumsum()
```

The syntax is obj.method(arguments here)

Input_data is an instance of a pd.DataFrame, and thus inherits its methods.
One method is .rename (to remain the columns)

The names are coming from doc

NORMAL main createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]


# change column names based on yaml doc
input_data.rename(columns=doc["columnNa

# create running timer
input_data["timing"] = input_data["time

# create violation detection
input_data["violation"] = np.where(inpu

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
aseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

Objects can have methods associated with them… methods are like functions but they take the object as the first argument.

```
# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time"].cumsum()
```

The syntax is obj.method(arguments here)

Input_data is an instance of a pd.DataFrame, and thus inherits its methods.
One method is .rename (to remain the columns)

The names are coming from doc

NORMAL main createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]

# change column names based on yaml doc
input_data.rename(columns=doc["columnNa

# create running timer
input_data["timing"] = input_data["time

# create violation detection
input_data["violation"] = np.where(inpu

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
aseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

Objects can have methods associated with them… methods are like functions but they take the object as the first argument.

```
# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time"].cumsum()
```

columns is an argument to the .rename method

NORMAL main createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]

# change column names based on yaml doc
input_data.rename(columns=doc["columnNa

# create running timer
input_data["timing"] = input_data["time

# create violation detection
input_data["violation"] = np.where(inpu

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
aseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

Objects can have methods associated with them… methods are like functions but they take the object as the first argument.

```
# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time"].cumsum()
```

inplace is also an argument to the .rename method

NORMAL main⚡ createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]

# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time
```

Adding a new column named “violation” to the data frame input_data

```
# create violation detection
input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
aseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```


```
# create violation detection
input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)
```


NORMAL main⚡ createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]

# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time

# create violation detection
input_data["violation"] = np.where(inpu

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
aseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

Adding a new column named “violation” to the data frame input_data

```
# create violation detection
input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
```

np was imported, numpy

np.where is a function that acts like a ternary but is cleaner for creating a vector/column of values

NORMAL main createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

vim createPlots.py

```
# import constants from yaml doc
if TYPE is True:
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
elif TYPE is False:
    NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
else:
    print("No type given as third argument, assuming property line")
    NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
COLOR_SCALE = doc["colorScale"]
Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
if TYPE is True:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
elif TYPE is False:
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
else:
    print("No type given as third argument, assuming property line")
    Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
VIOLATION_T_MARK = doc["violationTFMark"][0]
VIOLATION_F_MARK = doc["violationTFMark"][1]

# change column names based on yaml doc
input_data.rename(columns=doc["columnNames"], inplace=True)

# create running timer
input_data["timing"] = input_data["time

# create violation detection
input_data["violation"] = np.where(inpu

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
baseChart = (
    alt.Chart(input_data)
    .mark_line(color="red")
    .encode(
        alt.X("timing", scale=alt.Scale(zero=False)),
```

Adding a new column named “violation” to the data frame input_data

```
# create violation detection
input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)

# print
print(TYPE)
print(NOISE_VIOLATION_THRESHOLD)
print(Y_SCALE_DOMAIN_MAX)

# begin time series plot
```

np was imported, numpy

np.where is a function that acts like a ternary but is cleaner for creating a vector/column of values

np.where(CONDITION, True_Return_This, False_Return_This

NORMAL main createPlots.py python utf-8[unix] 33% 42/126 :1 W:10(L1) E:2(L72)

vim createPlots.py

```
42
 1 # import constants from yaml doc
 2 if TYPE is True:
 3     NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
 4 elif TYPE is False:
 5     NOISE_VIOLATION_THRESHOLD = doc["noise75FeetThreshold"]
 6 else:
 7     print("No type given as third argument, assuming property line")
 8     NOISE_VIOLATION_THRESHOLD = doc["noisePropertyLineThreshold"]
 9 COLOR_SCALE = doc["colorScale"]
10 Y_SCALE_DOMAIN_MIN = doc["yScaleDomainMin"]
11 if TYPE is True:
12     Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
13 elif TYPE is False:
14     Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"] - doc["yScaleAdjust75Feet"]
15 else:
16     print("No type given as third argument, assuming property line")
17     Y_SCALE_DOMAIN_MAX = doc["yScaleDomainMax"]
18 COLOR_SCALE_DOMAIN_MIN = doc["colorScaleDomainMin"]
19 COLOR_SCALE_DOMAIN_MAX = doc["colorScaleDomainMax"]
20 VIOLATION_T_MARK = doc["violationTFMark"][0]
21 VIOLATION_F_MARK = doc["violationTFMark"][1]
22
23 # change column names based on yaml doc
24 input_data.rename(columns=doc["columnNames"], inplace=True)
25
26 # create running timer
27 input_data["timing"] = input_data["time
28
29 # create violation detection
30 input_data["violation"] = np.where(inpu
31
32 # print
33 print(TYPE)
34 print(NOISE_VIOLATION_THRESHOLD)
35 print(Y_SCALE_DOMAIN_MAX)
36
37 # begin time series plot
38 aseChart = (
39     alt.Chart(input_data)
40     .mark_line(color="red")
41     .encode(
42         alt.X("timing", scale=alt.Scale(zero=False)),
```

Adding a new column named “violation” to the data frame input_data

```
29 # create violation detection
30 input_data["violation"] = np.where(input_data["dBA"] > NOISE_VIOLATION_THRESHOLD, True, False)
31
32 # print
33 print(TYPE)
34 print(NOISE_VIOLATION_THRESHOLD)
35 print(Y_SCALE_DOMAIN_MAX)
36
37 # begin time series plot
```

np was imported, numpy

np.where is a function that acts like a ternary but is cleaner for creating a vector/column of values

np.where(CONDITION, True_Return_This, False_Return_This

So this fills in the values for the new column “violation” in input_data

NORMAL main⚡ createPlots.py python utf-8[unix] 33% ≡ 42/126 ㏑:1 W:10(L1) E:2(L72)

Added a few more accomplishments

Dictionaries {key: val, key: val}

pd.DataFrame()

methods, like pd.DataFrame().rename()

Create new columns myDataframe[“newColumn”] = …

Use np.where to create vectors of information

Going a little deeper

- Namespaces — when you same an object in memory, it has an address.
  - If something is in the global namespace, you can just type its name (like “if”)
  - Other objects are in other namespaces, for example argv is in the sys namespace, that is why you have to type sys.argv()
  - Namespaces avoid collisions between names that refer to different objects

Functions

- Functions:
  - It is useful to write your own functions
  - These are just recipes with inputs and outputs
  - In python we write:

```
def my_func(args, …):
    Do something
    return Something (or not)
```

Methods are functions that “belong” to a class of object… more on that next week

if __name__ == “__main__”

- When you look at other peoples code you will see this a lot
  - A condition looking for whether some variable that seems never to be set (__name__) is equal to “__main__”
- __name__ is set automatically Python.
  - When you are running a script, say createPlots.py; by typing python createPlots.py … at the command line, createPlots.py is the MAIN program.
  - __name__ is set equal to “__main__”
- But other times your code is not run … IT IS IMPORTED, import …
  - In that case __name__ is not main, but the name of the “module” imported

if __name__ == “__main__”

- This is used so that you can write code that both runs tests or outputs some values, but also can be imported in OTHER code
  - If you run the script… __name__ == “__main__” and so the lines of code run… doing things!
  - If you import the script __name__ != “__main__” and only the functions and class definitions are imported
    - There are in the … NAME space!

Example

```
def question1():
    print(42)
def question2():
    print(“True”)
def main():
    question1()
    question2()
If __name__ == “__main__”:
    main()
```