NIAPythonDay2

January 27, 2021

1 NIA Python Bootcamp Day 1 review

1. Python ecosystem of tools
2. Jupyter Notebook is code, output and documentation all in one document
3. Type code into cells, and to run them you press Shift-Enter
4. Different data types for different data
5. Tab completion reduces typing, shows you pop-up menu of all the things you can do with that piece of data
6. Operators take one or more input values and turn them into other values *based on the input values type*
7. Converting data from one type to another using the function syntax, e.g., int()

2 Day 2: Drilling down into data

2.1 Agenda for today

1. Exploring data types using the TAB key
2. Python syntax for taking slices of iterables
3. In-depth: NumPy arrays
4. Preview PANDAS DataFrames

2.2 Exploring a data type's built-in functionality

Here is some nucleotide data. The FASTA header for this file is:

`>ENA|AAC37009|AAC37009.1 Escherichia coli RNA helicase-like protein`

[1]:
```
nt = """ATGGCAGATAATCCAGACCCTTCATCGCTCCTGCCGGACGTGTTTTCACCGGCGACCCGC
GACTGGTTTCTTCGCGCCTTTAAACAGCCGACCGCTGTCCAGCCGCAAACCTGGCATGTG
GCGGCGCGAAGCGAACATGCGCTGGTGATTGCACCGACCGGCTCCGGGAAAACGCTGGCA
GCATTTCTCTACGCCCTCGATCGGCTCTTCCGCGAAGGCGGCGAAGATACCCGCGAGGCG
CATAAGCGTAAAACCTCACGCATCCTCTATATTTCACCGATAAAAGCCCTGGGCACCGAC
GTTCAGCGCAACTTGCAGATCCCGTTGAAGGGTATTGCCGATGAACGGCGGCGGCGCGGC
GAAACGGAAGTCAATCTTCGCGTAGGGATCCGTACTGGCGATACGCCTGCACAGGAACGC
AGCAAACTCACCCGTAATCCGCCGGATATTCTGATCACCACACCCGAATCACTCTATCTG
ATGCTGACCTCCCGCGCGCGCGAAACGCTACGCGGCGTCGAAACGGTAATTATTGATGAA
GTCCACGCGGTAGCGGGCAGTAAACGTGGTGCGCATCTGGCGTTAAGTCTGGAGCGGCTC
GATGCGCTGCTCCACACCTCAGCACAGCGAATTGGCCTTTCTGCCACTGTGCGCTCAGCC
AGCGATGTGGCAGCATTTCTTGGTGGCGATCGCCCGGTTACGGTAGTCAACCCGCCCGCA
```

```
ATGCGCCATCCGCAGATACGAATTGTCGTACCGGTCGCCAATATGGATGATGTCTCATCG
GTCGCCAGCGGCACCGGCGAAGACAGCCATGCCGGCCGGGAAGGCTCCATCTGGCCATAT
ATTGAAACGGGTATCCTTGATGAAGTGTTGCGCCATCGCTCGACCATTGTCTTTACTAAT
TCGCGGGGGCTGGCGGAAAAACTGACGGCACGATTAAATGAGCTTTACGCCGCACGCTTA
CAGCGTTCCCCGTCTATCGCCGTTGATGCGGCCCATTTCGAGTCGACCTCCGGCGCAACC
TCTAACCGTGTACAAAGTAGCGACGTTTTTATTGCCCGCTCACACCACGGCTCCGTCTCT
AAAGAACAACGAGCAATCACCGAACAGGCGCTGAAATCGGGTGAATTACGCTGCGTGGTC
GCAACCTCCAGTCTTGAACTGGGGATTGATATGGGCGCGGTGGATCTGGTGATTCAGGTG
GCAACGCCGCTTTCTGTTGCCAGTGGGTTACAACGCATTGGTCGCGCCGGACATCAGGTT
GGCGGTGTATCTAAAGGGCTGTTTTTCCCCCGTACCCGGCGTGATTTAGTCGATTCCGCA
GTCATTGTAGAGTGTATGTTCGCAGGCAGGCTGGAAAACCTGACACCACCGCATAATCCT
CTCGACGTCCTTGCGCAGCAAACCGTTGCCGCCGCGGCGATGGATGCATTACAGGTAGAC
GAATGGTACTCCCGCGTACGCCGTGCCGCACCGTGGAAAGATCTGCCAAGACGTGTTTTT
GACGCCACGCTGGATATGCTTTCCGGGCGCTATCCCTCTGGCGATTTTTCTGCTTTTCGC
CCCAAACTGGTCTGGAACAGGGAGACCGGGATATTGACCGCCCGACCTGGCGCTCAATTG
TTGGCGGTTACCAGCGGCGGCACCATTCCGGATCGTGGCATGTATAGCGTGTTATTACCC
GAAGGTGAAGAAAAGGCCGGTTCGCGGCGGGTGGGTGAACTGGATGAGGAGATGGTATAT
GAGTCGCGGGTGAACGACATTATCACTCTCGGCGCTACCTCATGGCGGATCCAGCAAATC
ACCCGCGATCAGGTGATTGTGACTCCTGCTCCGGGTCGTTCTGCCCGGCTCCCCTTCTGG
CGTGGTGAAGGTAACGGACGTCCGGCTGAATTAGGCGAGATGATCGGCGATTTTCTTCAT
TTGCTGGCGGATGGCGCGTTCTTTTCCGGGACTATTCCCCCGTGGCTGGCAGAAGAAAAT
ACGATCGCCAATATTCAGGGGTTGATTGAGGAGCAGCGCAACGCGACGGGCATCGTTCCG
GGGAGTCGCCATCTGGTCCTCGAACGGTGCCGTGATGAAATTGGTGACTGGCGTATTATT
TTGCACTCTCCCTATGGAAGACGGGTGCATGAACCCTGGGCGGTGGCGATTGCCGGGCGA
ATACATGCGCTATGGGGCGCTGACGCGTCGGTGGTCGCCAGTGATGACGGCATTGTTGCA
CGTATTCCTGACACCGATGGTAAATTGCCCGATGCCGCGATTTTTTTGTTTGAACCAGAA
AAGTTGCTGCAAATTGTCCGCGAGGCGGTAGGCAGCTCGGCACTTTTCGCCGCCCGTTTT
CGCGAATGCGCCGCGCGGGCATTATTAATGCCGGGGCGCACTCCGGGCCATCGCACCCCG
CTTTGGCAACAACGGCTGCGCGCCAGTCAGTTGCTGGAAATCGCTCAGGGATATCCGGAT
TTTCCGGTCATTCTCGAAACCCTACGCGAATGTCTGCAAGATGTTTATGATCTTCCCGCA
CTGGAACGTTTGATGCGTCGCCTGAACGGTGGCGAAATTCAAATATCCGATGTAACGACC
ACTACGCCCTCGCCTTTCGCCACAAGTTTATTGTTCGGCTATGTCGCGGAATTTATGTAC
CAGAGCGACGCCCCGCTGGCAGAGCGCCGGGCATCCGTACTGTCGCTGGACAGCGAGTTA
CTGCGCAATCTACTCGGACAGGTCGATCCGGGGGAATTACTCGACCCGCAGGTCATTCGC
CAGGTGGAAGAAGAGTTGCAACGACTGGCTCCTGGCAGAAGAGCGAAAGGTGAAGAAGGA
TTGTTCGACCTGCTGCGCGAACTGGGGCCAATGACCGTTGAAGACCTGGCGCAACGGCAT
ACAGGCAGCAGTGAAGAGGTTGCGTCGTATCTGGAAAATCTTCTTGCAGTAAAACGAATC
TTCCCAGCGATGATTAGCGGACAGGAGCGTCTTGCCTGTATGGATGATGCCGCCAGGCTG
CGTGATGCCCTCGGCGTACGACTACCAGAGTCATTGCCAGAGATTTATTTACATAGAGTC
AGTTACCCGCTTCGCGACCTCTTTCTGCGCTATCTCCGGGCTCATGCTCTGGTCACGGCT
GAACAACTGGCTCATGAGTTTAGTCTCGGTATTGCCATTGTCGAAGAGCAGCTTCAGCAA
CTGCGTGAACAGGGTCTGGTGATGAATCTGCAACAAGACATCTGGGTGAGCGATGAAGTA
TTTCGTCGTCTGCGTTTGCGCTCGCTGCAAGCCGCCAGAGAAGCGACGCGTCCCGTTGCA
GCCACGACCTATGCGCGATTGCTGCTGGAACGTCAGGGCGTATTACCCGCCACCGATGGT
AGCCCGGCGCTCTTTGCCTCAACATCGCCAGGCGTTTATGAGGGCGTAGATGGCGTGATG
CGGGTGATCGAACAGCTTGCCGGAGTCGGTTTACCCGCCTCACTCTGGGAAAGCCAGATC
CTGCCTGCCCGCGTACGCGACTATTCGTCAGAAATGCTCGATGAATTACTGGCAACCGGT
```

```
GCGGTTATCTGGTCGGGGCAAAAAAAACTGGGTGAAGATGACGGCCTGGTGGCACTGCAT
CTACAGGAATATGCTGCAGAATCGTTCACTCCCGCCGAAGCGGATCAGGCGAATCGTTCG
GCGCTGCAACAAGCGATAGTCGCTGTTCTGGCTGACGGAGGAGCCTGGTTTGCACAACAA
ATCAGCCAACGGATACGCGACAAAATCGGCGAATCGGTTGATCTCTCTGCCCTGCAAGAG
GCGTTATGGGCGCTGGTCTGGCAAGGCGTCATCACCAGCGACATTTGGGCACCGTTACGC
GCCCTCACCCGCAGCAGTTCCAACGCACGCACCTCAACTCGCCGCAGTCACCGGGCTCGT
CGTGGACGTCCTGTCTATGCGCAACCCGTCTCGCCGCGGGTATCTTACAACACACCAAAT
CTGGCTGGACGCTGGTCGTTATTGCAGGTGGAGCCACTAAACGATACCGAAAGGATGCTG
GCGCTGGCGGAAAATATGCTCGACCGCTACGGCATCATCAGTCGTCAGGCGGTGATAGCC
GAAAATATCCCTGGCGGGTTTCCATCGATGCAAACGCTTTGTCGAAGTATGGAAGACTCC
GGGCGAATTATGCGAGGTCGTTTTGTAGAAGGTCTGGGTGGCGCGCAATTCGCTGAACGT
CTGACTATTGACCGATTGCGCGATCTGGCGACACAAGCCACGCAAACGCGCCACTATACA
CCAGTGGCGCTCTCTGCCAACGATCCGGCTAATGTGTGGGGAAATCTTCTGCCCTGGCCT
GCACATCCGGCAACGCTGGTTCCAACGCGTCGGGCGGGTGCGCTGGTGGTCGTTTCTGGC
GGCAAATTGTTACTCTATCTGGCGCAAGGTGGCAAAAAAATGCTGGTCTGGCAGGAAAAA
GAGGAATTACTCGCCCCAGAGGTTTTCCACGCGCTGACTACCGCACTGCGTCGCGAACCA
CGGCTGCGCTTTACGCTAACAGAAGTGAATGATCTACCGGTCCGGCAAACGCCGATGTTT
ACGCTGCTGCGTGAGGCGGGATTTTCAAGTTCGCCACAAGGGCTGGATTGGGGATAG
"""
```

[2]:
```
len( nt )
```

[2]: 4694

[Entry in NCBI Protein database](#) says protein is 1,538 amino acids long (4,614 nucleotides)

[3]:
```
len( nt ) / 3
```

[3]: 1564.6666666666667

Why? Answer: Escape Characters

[ ]:
```
nt
```

[ ]:
```
print( nt )
```

[ ]:
```
nt.replace?
```

[4]:
```
len( nt )
```

[4]: 4694

[5]:
```
nt = nt.replace( "\n", "")
```

[6]:
```
len( nt )
```

[6]: 4617

[7]:
```
len( nt ) / 3
```

[7]: 1539.0

2.2.1 What type of data does the name nt point to?

[8]:
```
type( nt )
```

[8]: str

2.2.2 How many nucleotides in this gene? Use len()

[9]:
```
len(nt)
```

[9]: 4617

2.2.3 How many amino acids in this gene?

[10]:
```
len(nt)/3
```

[10]: 1539.0

2.2.4 Use the tab key to see what else you can do with this string

- The pop-up menu with show a list of *attributes* associated with that value.
- Attributes is the technical term for all the functions and metadata that is attached to a value.
- Use the question mark if you want to learn more about an attribute and what it does

[ ]:
```
nt.
```

2.2.5 Use the .count() function that's built into most iterables to measure GC content %

[11]:
```
"GGGGGGGGGG".count('G')
```

[11]: 10

[12]:
```
"GCGCGCGCGC".count('G')
```

[12]: 5

[13]:
```
"GCGCGCGCGC".count('GC')
```

[13]: 5

[14]:
```
"GGGGG".count('GG')
```

[14]: 2

[ ]:
```
nt.count?
```

```
[15]: nt.count( "GC")
```

```
[15]: 458
```

```
[16]: (nt.count('G') + nt.count('C')) / len(nt)
```

```
[16]: 0.5689841888672298
```

2.3 Slicing Iterables

You can use the slice notation on lists, strings and more.

```
[17]: alphabet_str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

```
[18]: alphabet_str
```

```
[18]: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

2.3.1 Subsets: slicing iterables into smaller ones

Return a substring using a brackets separated by a colon.

```
[19]: len(alphabet_str)
```

```
[19]: 26
```

2.3.2 Slicing an iterable doesn't change the original iterable

Just because you just returned a substring from a string doesn't mean you changed the original string. Python created a new string and returned that

```
[20]: len(alphabet_str)
```

```
[20]: 26
```

```
[21]: alphabet_str[0:10]
```

```
[21]: 'ABCDEFGHIJ'
```

```
[22]: len(alphabet_str)
```

```
[22]: 26
```

2.3.3 Slicing syntax

[begin index : end index : step]

- ***NOTE PYTHON SLICING CONVENTION:*** Iterable indices start from 0!!!!
- ***NOTE ANOTHER PYTHON SLICING CONVENTION:*** The begin index is inclusive, the end index is exclusive!!!!!

[23]:
```python
alphabet_str[0]
```

[23]: 'A'

[24]:
```python
alphabet_str[25]
```

[24]: 'Z'

[25]:
```python
alphabet_str[26]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-25-81830bf77d10> in <module>
----> 1 alphabet_str[26]

IndexError: string index out of range
```

[26]:
```python
alphabet_str[0:3]
```

[26]: 'ABC'

[27]:
```python
alphabet_str
```

[27]: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

The following translates to: "give me the slice from the 0th index inclusive, to the 26th index exclusive (i.e. the 25th index)

[28]:
```python
alphabet_str[26]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-28-81830bf77d10> in <module>
----> 1 alphabet_str[26]

IndexError: string index out of range
```

[29]:
```python
# This gives the whole alphabet:
alphabet_str[0:26]
```

[29]: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

[30]:
```python
# This one leaves off the letter at the 25th index, which counting from 0 is Z:
alphabet_str[0:25]
```

[30]: 'ABCDEFGHIJKLMNOPQRSTUVWXY'

[31]:
```python
alphabet_str[::2] # take every 3rd letter
```

[31]: 'ACEGIKMOQSUWY'

[32]:
```python
alphabet_str[1:17:2] # take every 3rd letter
```

[32]: 'BDFHJLNP'

[33]:
```python
alphabet_str[1:26:3]
```

[33]: 'BEHKNQTWZ'

3 Exercise: Iterating over all the "codons" in the alphabet.

Fill in the code to achieve the following output.

```
ABC
DEF
GHI
JKL
MNO
PQR
STU
VWX
YZ
```

Pseudocode: * Initialize a positional variable that keeps track of where you are in the string * `while` the current position is less than the full length of the string * Slice the full string into a substring beginning at the current position and ending three letters later * Print out the substring * Increment the positional variable by 3

[ ]:
```python
# Fill in your code here:
```

3.0.1 Slice from the beginning to the middle somewhere

Leave out the start index and Python assumes you want a slice starting from the beginning.

[34]:
```python
alphabet_str[ :8]
```

[34]: 'ABCDEFGH'

3.0.2 Slice from the middle somewhere to the end

Leave out the end index and Python assumes you want a slice that goes straight to the end.

[35]:
```python
alphabet_str[20:]
```

[35]: 'UVWXYZ'

3.0.3 Negative slice indices mean count from the end

If i is negative, index is relative to end of string:

[36]:
```
alphabet_str[-25:]
```

[36]: 'BCDEFGHIJKLMNOPQRSTUVWXYZ'

[37]:
```
alphabet_str[1:]
```

[37]: 'BCDEFGHIJKLMNOPQRSTUVWXYZ'

[38]:
```
alphabet_str[::-1]
```

[38]: 'ZYXWVUTSRQPONMLKJIHGFEDCBA'

3.0.4 Reverse a the order of an iterable using the step parameter

Reverse a string by using a negative step value

[39]:
```
"a man, a plan, a canal, panama"[::-1]
```

[39]: 'amanap ,lanac a ,nalp a ,nam a'

[40]:
```
test = "a man, a plan, a canal, panama"
```

[41]:
```
test
```

[41]: 'a man, a plan, a canal, panama'

3.0.5 Star operator for lists

- For lists, the * repeats the list, not element-wise multiply

[42]:
```
numbers = [ 1,2,3,4,5 ]
```

[43]:
```
numbers
```

[43]: [1, 2, 3, 4, 5]

[44]:
```
numbers * 2
```

[44]: [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

[45]:
```
# Element-wise using plain vanilla python
# using a "list comprehension"
[ num * 2 for num in numbers ]
```

[45]: [2, 4, 6, 8, 10]

3.1 NumPy arrays

- Use for 1-D, 2-D, n-D data
- Use if you have data all of the same type (ints, floats, bools)

[46]:
```
# import the package and give it a nickname "np" for short
import numpy as np
```

3.1.1 Make a 1-D NumPy array from a list

[47]:
```
a_list = [1,2,3,4,5,6,7,8,9,10]
```

[48]:
```
a_list
```

[48]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[49]:
```
type( a_list)
```

[49]:
```
list
```

[50]:
```
an_array = np.array( a_list )
```

[51]:
```
an_array
```

[51]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[52]:
```
type( an_array)
```

[52]:
```
numpy.ndarray
```

3.1.2 Get the length of the NumPy array

[53]:
```
len( an_array )
```

[53]:
```
10
```

3.1.3 Slice 1-D NumPy Arrays the same way you slice built-in Python iterables

[54]:
```
an_array[ 5:]
```

[54]:
```
array([ 6,  7,  8,  9, 10])
```

3.1.4 NumPy arrays have many basic statistics and functions built-in

Use the .TAB trick to get the pop-up menu to see your options, and use the question mark to see each attribute's documentation.

[55]:
```
an_array.max()
```

[55]: 10

[56]:
```
an_array.mean()
```

[56]: 5.5

[57]:
```
an_array.min()
```

[57]: 1

3.1.5 Use index notation to change values in an array

[58]:
```
the_slice = an_array[ 4:7 ]
```

[59]:
```
the_slice
```

[59]: array([5, 6, 7])

[60]:
```
an_array
```

[60]: array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

[61]:
```
an_array[4] = 42
```

[62]:
```
an_array
```

[62]: array([1, 2, 3, 4, 42, 6, 7, 8, 9, 10])

3.1.6 Matrix math: Multiply/divide by a constant

[63]:
```
an_array * 10
```

[63]: array([10, 20, 30, 40, 420, 60, 70, 80, 90, 100])

[64]:
```
an_array
```

[64]: array([1, 2, 3, 4, 42, 6, 7, 8, 9, 10])

[65]:
```
an_array = an_array * 10
```

[66]:
```
an_array *= 10
```

3.1.7 Matrix math: add/subtract constant

[67]:
```
an_array
```

[67]: array([100, 200, 300, 400, 4200, 600, 700, 800, 900, 1000])

[68]:
```python
an_array - 1
```

[68]:
```
array([  99,  199,  299,  399, 4199,  599,  699,  799,  899,  999])
```

3.1.8 Matrix math: Z-Score normalization

1. Subtract the mean value (scalar) from all values
2. Divide all_values by the standard deviation (scalar)

[69]:
```python
# start with a fresh dataset
a_list = [1,2,3,4,5,6,7,8,9,10]
an_array = np.array( a_list )
```

[70]:
```python
an_array.mean()
```

[70]:
```
5.5
```

[71]:
```python
an_array.std()
```

[71]:
```
2.8722813232690143
```

[72]:
```python
an_array - an_array.mean() / an_array.std()
```

[72]:
```
array([-0.91485422,  0.08514578,  1.08514578,  2.08514578,  3.08514578,
        4.08514578,  5.08514578,  6.08514578,  7.08514578,  8.08514578])
```

3.1.9 Subselect based on a boolean criterion

Give me all values that are greater than a certain value

[73]:
```python
an_array
```

[73]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[74]:
```python
an_array > 5
```

[74]:
```
array([False, False, False, False, False,  True,  True,  True,  True,
        True])
```

[75]:
```python
an_array[ an_array > 5]
```

[75]:
```
array([ 6,  7,  8,  9, 10])
```

Give me all even numbers in this array, using the Modulus division operator %

[76]:
```python
an_array % 2
```

[76]:
```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 0])
```

[77]:
```python
an_array % 2 == 0
```

[77]: array([False, True, False, True, False, True, False, True, False,
        True])

[78]:
```
an_array[ an_array % 2 == 0 ]
```

[78]: array([2, 4, 6, 8, 10])

3.2 2-D NumPy Arrays

3.2.1 Use NumPy's arange() function to quickly generate a list of counting numbers

[79]:
```
new_array = np.arange(120)
```

[80]:
```
new_array
```

[80]:
```
array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
       104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
       117, 118, 119])
```

[81]:
```
len(new_array)
```

[81]: 120

[82]:
```
new_array.shape
```

[82]: (120,)

[83]:
```
new_array.mean()
```

[83]: 59.5

3.2.2 Use the .reshape() function to convert a 1-D array into 2-D

- Use (num_rows, num_cols) notation

[84]:
```
new_array = new_array.reshape( (20, 6) )
```

[85]:
```
new_array
```

[85]:
```
array([[  0,   1,   2,   3,   4,   5],
       [  6,   7,   8,   9,  10,  11],
       [ 12,  13,  14,  15,  16,  17],
```

```
      [ 18,  19,  20,  21,  22,  23],
      [ 24,  25,  26,  27,  28,  29],
      [ 30,  31,  32,  33,  34,  35],
      [ 36,  37,  38,  39,  40,  41],
      [ 42,  43,  44,  45,  46,  47],
      [ 48,  49,  50,  51,  52,  53],
      [ 54,  55,  56,  57,  58,  59],
      [ 60,  61,  62,  63,  64,  65],
      [ 66,  67,  68,  69,  70,  71],
      [ 72,  73,  74,  75,  76,  77],
      [ 78,  79,  80,  81,  82,  83],
      [ 84,  85,  86,  87,  88,  89],
      [ 90,  91,  92,  93,  94,  95],
      [ 96,  97,  98,  99, 100, 101],
      [102, 103, 104, 105, 106, 107],
      [108, 109, 110, 111, 112, 113],
      [114, 115, 116, 117, 118, 119]])
```

[86]:
```
new_array.shape
```

[86]:
```
(20, 6)
```

3.2.3 Get the mean for the whole matrix

[87]:
```
new_array.mean()
```

[87]:
```
59.5
```

3.2.4 Get the column wise mean

[88]:
```
new_array.mean( axis=0 )
```

[88]:
```
array([57., 58., 59., 60., 61., 62.])
```

3.2.5 Get the row-wise mean

[89]:
```
new_array.mean( axis=1 )
```

[89]:
```
array([  2.5,   8.5,  14.5,  20.5,  26.5,  32.5,  38.5,  44.5,  50.5,
        56.5,  62.5,  68.5,  74.5,  80.5,  86.5,  92.5,  98.5, 104.5,
       110.5, 116.5])
```

3.2.6 Transpose the array using the .T attribute

[90]:
```
new_array.T
```

[90]:
```
array([[  0,   6,  12,  18,  24,  30,  36,  42,  48,  54,  60,  66,  72,
         78,  84,  90,  96, 102, 108, 114],
```

```
       [  1,   7,  13,  19,  25,  31,  37,  43,  49,  55,  61,  67,  73,
         79,  85,  91,  97, 103, 109, 115],
       [  2,   8,  14,  20,  26,  32,  38,  44,  50,  56,  62,  68,  74,
         80,  86,  92,  98, 104, 110, 116],
       [  3,   9,  15,  21,  27,  33,  39,  45,  51,  57,  63,  69,  75,
         81,  87,  93,  99, 105, 111, 117],
       [  4,  10,  16,  22,  28,  34,  40,  46,  52,  58,  64,  70,  76,
         82,  88,  94, 100, 106, 112, 118],
       [  5,  11,  17,  23,  29,  35,  41,  47,  53,  59,  65,  71,  77,
         83,  89,  95, 101, 107, 113, 119]])
```

[91]:
```
new_array.T.shape
```

[91]:
```
(6, 20)
```

3.2.7 Zscore standardize by columns

[92]:
```
# Use numpy's set_printoptions to change display precision
np.set_printoptions( precision = 2)
```

[93]:
```
(new_array - new_array.mean(axis=0)) / new_array.std(axis=0)
```

[93]:
```
array([[-1.65, -1.65, -1.65, -1.65, -1.65, -1.65],
       [-1.47, -1.47, -1.47, -1.47, -1.47, -1.47],
       [-1.3 , -1.3 , -1.3 , -1.3 , -1.3 , -1.3 ],
       [-1.13, -1.13, -1.13, -1.13, -1.13, -1.13],
       [-0.95, -0.95, -0.95, -0.95, -0.95, -0.95],
       [-0.78, -0.78, -0.78, -0.78, -0.78, -0.78],
       [-0.61, -0.61, -0.61, -0.61, -0.61, -0.61],
       [-0.43, -0.43, -0.43, -0.43, -0.43, -0.43],
       [-0.26, -0.26, -0.26, -0.26, -0.26, -0.26],
       [-0.09, -0.09, -0.09, -0.09, -0.09, -0.09],
       [ 0.09,  0.09,  0.09,  0.09,  0.09,  0.09],
       [ 0.26,  0.26,  0.26,  0.26,  0.26,  0.26],
       [ 0.43,  0.43,  0.43,  0.43,  0.43,  0.43],
       [ 0.61,  0.61,  0.61,  0.61,  0.61,  0.61],
       [ 0.78,  0.78,  0.78,  0.78,  0.78,  0.78],
       [ 0.95,  0.95,  0.95,  0.95,  0.95,  0.95],
       [ 1.13,  1.13,  1.13,  1.13,  1.13,  1.13],
       [ 1.3 ,  1.3 ,  1.3 ,  1.3 ,  1.3 ,  1.3 ],
       [ 1.47,  1.47,  1.47,  1.47,  1.47,  1.47],
       [ 1.65,  1.65,  1.65,  1.65,  1.65,  1.65]])
```

3.3 Subselecting a 2-D array using slicing

- The syntax for slicing on a 2-D NumPy array is similar to 1-D, except you use a comma.
- Rows, then columns

[94]:
```
new_array = np.arange(25).reshape( (5, 5) )
```

[95]:
```
new_array
```

[95]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

[96]:
```
new_array[0,0]
```

[96]:
```
0
```

[97]:
```
new_array[3,3]
```

[97]:
```
18
```

[98]:
```
new_array[ 2:4, 2:4 ]
```

[98]:
```
array([[12, 13],
       [17, 18]])
```

3.3.1 Use the colon : to indicate all rows or all columns

[99]:
```
new_array[:, 2:4 ]
```

[99]:
```
array([[ 2,  3],
       [ 7,  8],
       [12, 13],
       [17, 18],
       [22, 23]])
```

3.4 Example Image data as a 3-D NumPy array

[100]:
```
# Use the Python package matplotlib to render images and output them directly␣
↪to Jupyter Notebook
%matplotlib inline
import matplotlib.pyplot as plt
```

[101]:
```
from skimage.data import astronaut
```

[102]:
```
image_data = astronaut()
```

3.4.1 An RGB image has three color channels corresponding to Red Green and Blue

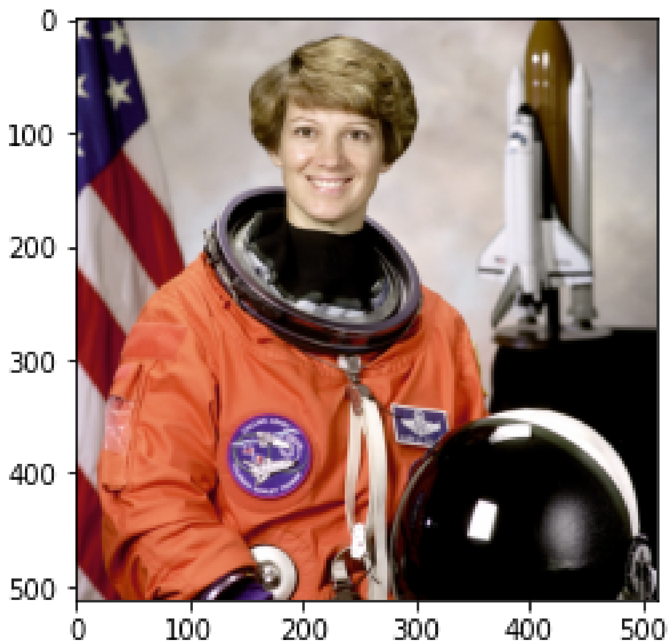
[103]:
```
image_data.shape
```

[103]:
```
(512, 512, 3)
```

3.4.2 Using matplotlib's imshow() function to see an image

[104]:
```
plt.imshow( image_data )
```

[104]:
```
<matplotlib.image.AxesImage at 0x101e00e50>
```



[105]:
```
image_data.max()
```

[105]:
```
255
```

[110]:
```
np.arange( 100 )
```

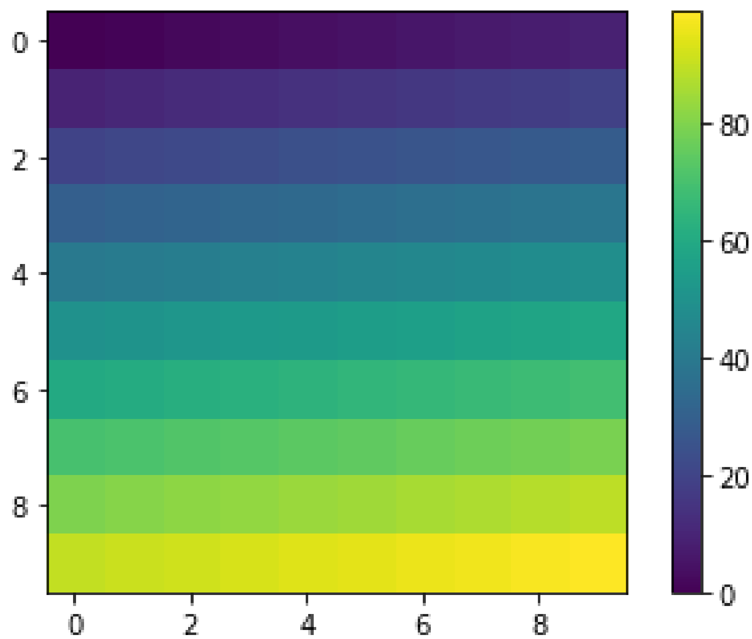
[110]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

[111]:
```
np.arange( 100 ).reshape( (10,10) )
```

[111]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

[109]:
```
test_img = np.arange( 100 ).reshape( (10,10) )
plt.imshow( test_img )
plt.colorbar()
```

[109]:
```
<matplotlib.colorbar.Colorbar at 0x14b629f70>
```



3.4.3 Subselect part of the image to show using slicing

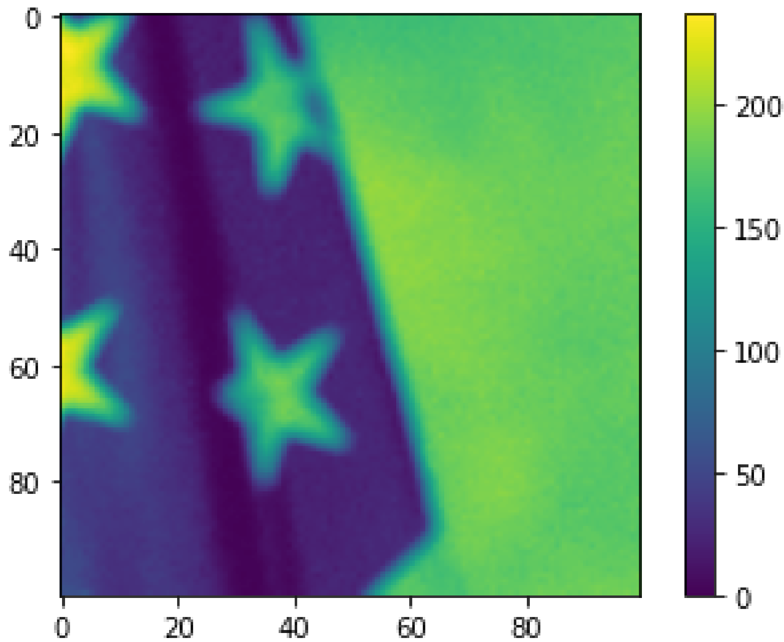
[113]:
```
image_data[ :100, :100, :]
```

[113]:
```
array([[[154, 147, 151],
        [109, 103, 124],
        [ 63,  58, 102],
        …,
        [174, 171, 174],
```

```
 [175, 172, 170],
 [175, 172, 171]],

[[177, 171, 171],
 [144, 141, 143],
 [113, 114, 124],
 …,
 [178, 175, 180],
 [175, 172, 171],
 [178, 175, 175]],

[[201, 194, 193],
 [182, 178, 175],
 [168, 165, 164],
 …,
 [177, 176, 180],
 [179, 177, 179],
 [177, 174, 174]],

…,

[[ 55,  40, 111],
 [ 53,  39, 109],
 [ 51,  38, 105],
 …,
 [184, 172, 168],
 [184, 176, 169],
 [183, 176, 170]],

[[ 58,  41, 116],
 [ 57,  44, 116],
 [ 55,  41, 112],
 …,
 [184, 177, 169],
 [182, 175, 166],
 [184, 174, 167]],

[[ 58,  42, 117],
 [ 59,  44, 119],
 [ 58,  44, 114],
 …,
 [189, 178, 173],
 [185, 175, 171],
 [187, 176, 174]]], dtype=uint8)
```

[114]:
```
plt.imshow( image_data[ :100, :100, 0] )
plt.colorbar()
```

[114]: <matplotlib.colorbar.Colorbar at 0x102d03be0>

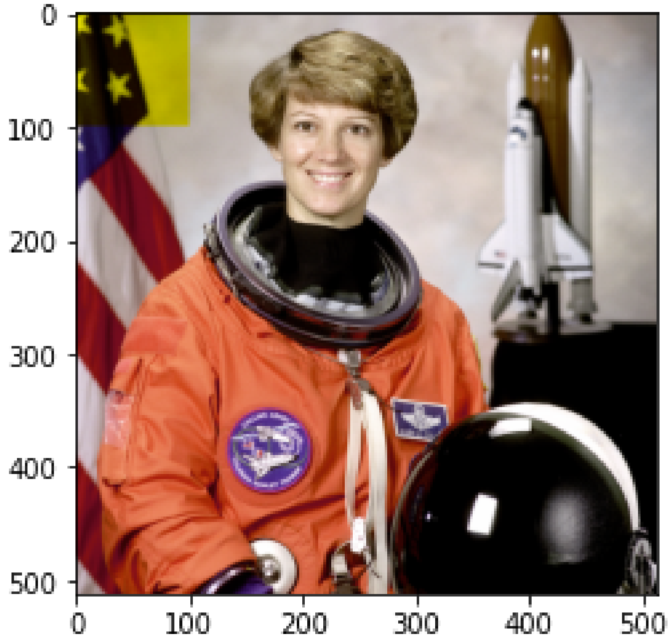


3.4.4 Zero out part of the blue channel

[115]:
```
image_data[ :100, :100, 2] = 0
```

[116]:
```
plt.imshow( image_data )
```

[116]: <matplotlib.image.AxesImage at 0x102d96610>



3.5 PANDAS DataFrame

- Emulate R's data.frame structure.
- Basically a NumPy matrix with
  - Row and column names
  - Can have columns of different types
  - Handles missing data better

[117]:
```
import pandas as pd
```

[118]:
```
titanic_data_url  = "https://gist.githubusercontent.com/michhar/
→2dfd2de0d4f8727f873422c5d959fff5/raw/
→fa71405126017e6a37bea592440b4bee94bf7b9e/titanic.csv"
```

[119]:
```
titanic = pd.read_csv( titanic_data_url )
```

[120]:
```
titanic.head()
```

[120]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
```

```
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[121]:
```
len( titanic )
```

[121]:
```
891
```

3.5.1 Change the number of rows Pandas will display using the set_option() function

Use the word None if you want to display all of them.

[122]:
```
pd.set_option( 'display.max_rows', None )
```