NIAPythonDay2

March 27, 2019

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`

```
In [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
"""
```

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

```
In [2]: nt = nt.replace( "\n", "")
```

```
In [ ]: nt
```

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

```
In [3]: type( nt )
```

```
Out[3]: str
```

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

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

```
Out[4]: 4617
```

2.2.3 How many amino acids in this gene?

```
In [5]: len(nt)/3
```

```
Out[5]: 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

```
In [ ]: nt.isalnum

In [ ]: nt.isalnum?

In [6]: nt.isalnum()

Out[6]: True

In [ ]: nt.count?

In [7]: nt.count('G')

Out[7]: 1344
```

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

```
In [8]: "GGGGGGGGGG".count('G')

Out[8]: 10

In [9]: "GCGCGCGCGC".count('G')

Out[9]: 5

In [10]: "GCGCGCGCGC".count('GC')

Out[10]: 5

In [11]: "GGGGG".count('GG')

Out[11]: 2

In [ ]: nt.count?

In [12]: nt.count( "GC")

Out[12]: 458

In [13]: (nt.count('G') + nt.count('C')) / len(nt)

Out[13]: 0.5689841888672298
```

2.3 Slicing Iterables

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

```
In [14]: a_string = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

In [15]: a_string

Out[15]: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

2.3.1 Subsets: slicing iterables into smaller ones

Return a substring using a brackets separated by a colon.

```
In [16]: len(a_string)
```

```
Out[16]: 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

```
In [17]: len(a_string)
```

```
Out[17]: 26
```

```
In [18]: a_string[0:10]
```

```
Out[18]: 'ABCDEFGHIJ'
```

```
In [19]: len(a_string)
```

```
Out[19]: 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!!!!!

```
In [20]: a_string[0]
```

```
Out[20]: 'A'
```

```
In [21]: a_string[25]
```

```
Out[21]: 'Z'
```

```
In [22]: a_string[26]
```

```
    ---------------------------------------------------------------------------

    IndexError                                Traceback (most recent call last)

    <ipython-input-22-c6bbde9f660a> in <module>
----> 1 a_string[26]


    IndexError: string index out of range
```

```
In [23]: a_string[0:3]
```

```
Out[23]: 'ABC'
```

```
In [24]: a_string
```

```
Out[24]: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

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

```
In [25]: a_string[26]
```

```
    ---------------------------------------------------------------------------

    IndexError                                Traceback (most recent call last)

    <ipython-input-25-c6bbde9f660a> in <module>
----> 1 a_string[26]


    IndexError: string index out of range
```

```
In [26]: # This gives the whole alphabet:
         a_string[0:26]
```

```
Out[26]: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
In [27]: # This one leaves off the letter at the 25th index, which counting from 0 is Z:
         a_string[0:25]
```

```
Out[27]: 'ABCDEFGHIJKLMNOPQRSTUVWXY'
```

```
In [28]: a_string[::2] # take every 3rd letter
```

```
Out[28]: 'ACEGIKMOQSUWY'
```

```
In [29]: a_string[1:17:2] # take every 3rd letter
```

```
Out[29]: 'BDFHJLNP'
```

```
In [30]: a_string[1:26:3]
```

```
Out[30]: 'BEHKNQTWZ'
```

3 Exercise

Write a code that prints the following output:

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

In [31]:
```
begin_index = 0
while begin_index < len( a_string ):
    end_index = begin_index + 3
    print( a_string[ begin_index : end_index ])
    begin_index += 3
```

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

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.

In [32]:
```
a_string[ :8]
```

Out[32]: '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.

In [33]:
```
a_string[20:]
```

Out[33]: 'UVWXYZ'

3.0.3 Negative slice indices mean count from the end

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

```
In [34]: a_string[-25:]
```

```
Out[34]: 'BCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
In [35]: a_string[1:]
```

```
Out[35]: 'BCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
In [36]: a_string[::-1]
```

```
Out[36]: 'ZYXWVUTSRQPONMLKJIHGFEDCBA'
```

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

Reverse a string by using a negative step value

```
In [37]: "a man, a plan, a canal, panama"[::-1]
```

```
Out[37]: 'amanap ,lanac a ,nalp a ,nam a'
```

```
In [38]: test = "a man, a plan, a canal, panama"
```

```
In [39]: test
```

```
Out[39]: 'a man, a plan, a canal, panama'
```

```
In [ ]: nt[::-1]
```

3.0.5 Star operator for lists

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

```
In [41]: numbers = [ 1,2,3,4,5]
```

```
In [42]: numbers
```

```
Out[42]: [1, 2, 3, 4, 5]
```

```
In [ ]: numbers * 2
```

```
In [43]: # Element-wise using plain vanilla python
         # using a "list comprehension"
         [ num * 2 for num in numbers ]
```

```
Out[43]: [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)

```
In [44]: # 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

```
In [45]: a_list = [1,2,3,4,5,6,7,8,9,10]

In [46]: a_list

Out[46]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [47]: type( a_list)

Out[47]: list

In [48]: an_array = np.array( a_list )

In [49]: an_array

Out[49]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])

In [50]: type( an_array)

Out[50]: numpy.ndarray
```

3.1.2 Get the length of the NumPy array

```
In [51]: len( an_array )

Out[51]: 10
```

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

```
In [53]: an_array[ 5:]

Out[53]: 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.

```
In [54]: an_array.max()

Out[54]: 10

In [55]: an_array.mean()

Out[55]: 5.5

In [56]: an_array.min()

Out[56]: 1
```

3.1.5 Use index notation to change values in an array

```
In [57]: the_slice = an_array[ 4:7 ]

In [58]: the_slice

Out[58]: array([5, 6, 7])

In [59]: an_array

Out[59]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])

In [60]: an_array[4] = 42

In [61]: an_array

Out[61]: array([ 1,  2,  3,  4, 42,  6,  7,  8,  9, 10])
```

3.1.6 Matrix math: Multiply/divide by a constant

```
In [62]: an_array * 10

Out[62]: array([ 10,  20,  30,  40, 420,  60,  70,  80,  90, 100])

In [63]: an_array

Out[63]: array([ 1,  2,  3,  4, 42,  6,  7,  8,  9, 10])

In [64]: an_array = an_array * 10

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

3.1.7 Matrix math: add/subtract constant

```
In [66]: an_array

Out[66]: array([ 100,  200,  300,  400, 4200,  600,  700,  800,  900, 1000])

In [67]: an_array - 1

Out[67]: 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)

```
In [68]: # start with a fresh dataset
         a_list = [1,2,3,4,5,6,7,8,9,10]
         an_array = np.array( a_list )

In [69]: an_array.mean()
```

```
Out[69]: 5.5

In [70]: an_array.std()

Out[70]: 2.8722813232690143

In [71]: an_array - an_array.mean() / an_array.std()

Out[71]: 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

```
In [72]: an_array

Out[72]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])

In [73]: an_array > 5

Out[73]: array([False, False, False, False, False,  True,  True,  True,  True,
                 True])

In [74]: an_array[ an_array > 5]

Out[74]: array([ 6,  7,  8,  9, 10])
```

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

```
In [75]: an_array % 2

Out[75]: array([1, 0, 1, 0, 1, 0, 1, 0, 1, 0])

In [76]: an_array % 2 == 0

Out[76]: array([False,  True, False,  True, False,  True, False,  True, False,
                 True])

In [77]: an_array[ an_array % 2 == 0 ]

Out[77]: 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

```
In [78]: new_array = np.arange(120)

In [79]: new_array
```

```
Out[79]: 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])
```

```
In [80]: len(new_array)
```

```
Out[80]: 120
```

```
In [81]: new_array.shape
```

```
Out[81]: (120,)
```

```
In [82]: new_array.mean()
```

```
Out[82]: 59.5
```

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

- Use (num_rows, num_cols) notation

```
In [83]: new_array = new_array.reshape( (20, 6) )
```

```
In [84]: new_array
```

```
Out[84]: 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]])
```

```
In [85]: new_array.shape
```

```
Out[85]: (20, 6)
```

3.2.3 Get the mean for the whole matrix

```
In [86]: new_array.mean()
```

```
Out[86]: 59.5
```

3.2.4 Get the column wise mean

```
In [87]: new_array.mean( axis=0 )
```

```
Out[87]: array([57., 58., 59., 60., 61., 62.])
```

3.2.5 Get the row-wise mean

```
In [88]: new_array.mean( axis=1 )
```

```
Out[88]: 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

```
In [89]: new_array.T
```

```
Out[89]: 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]])
```

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

```
Out[90]: (6, 20)
```

3.2.7 Zscore standardize by columns

In [91]:
```
# Use numpy's set_printoptions to change display precision
np.set_printoptions( precision = 2)
```

In [92]:
```
(new_array - new_array.mean(axis=0)) / new_array.std(axis=0)
```

Out[92]:
```
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

In [93]:
```
new_array = np.arange(25).reshape( (5, 5) )
```

In [94]:
```
new_array
```

Out[94]:
```
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]])
```

In [95]:
```
new_array[0,0]
```

Out[95]:
```
0
```

In [96]:
```
new_array[3,3]
```

Out[96]:
```
18
```

```
In [97]: new_array[ 2:4, 2:4 ]

Out[97]: array([[12, 13],
                [17, 18]])
```

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

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

Out[98]: array([[ 2,  3],
                [ 7,  8],
                [12, 13],
                [17, 18],
                [22, 23]])
```

3.4 Example Image data as a 3-D NumPy array

```
In [99]: # Use the Python package matplotlib to render images and output them directly to Jupy
         %matplotlib inline
         import matplotlib.pyplot as plt

In [100]: from skimage.data import astronaut

In [101]: image_data = astronaut()
```

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

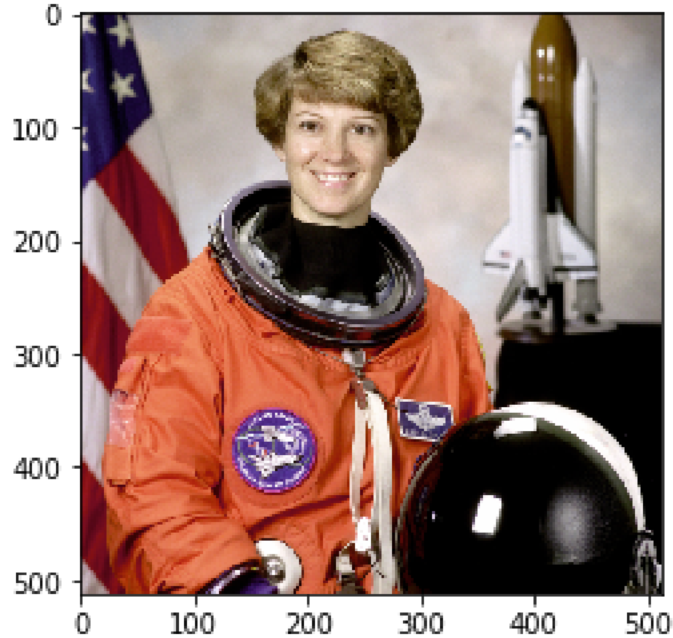
```
In [102]: image_data.shape

Out[102]: (512, 512, 3)
```

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

```
In [103]: plt.imshow( image_data )

Out[103]: <matplotlib.image.AxesImage at 0x1265f89b0>
```



```
In [104]: image_data.max()
```

```
Out[104]: 255
```

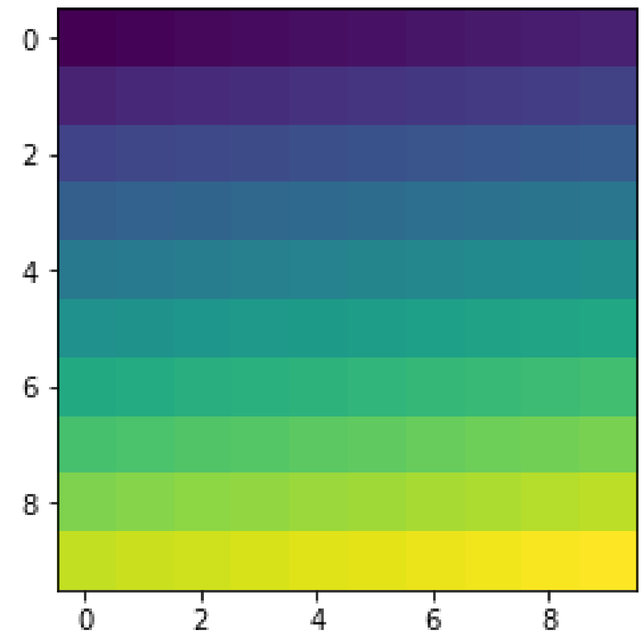
```
In [105]: #test_img = np.zeros(( 10,10))
          test_img = np.ones(( 10,10)) * 255
```

```
In [106]: test_img[0,0] = 0
```

```
In [107]: test_img = np.arange( 100 ).reshape( (10,10) )
```

```
In [108]: plt.imshow( test_img )
```

```
Out[108]: <matplotlib.image.AxesImage at 0x126699780>
```

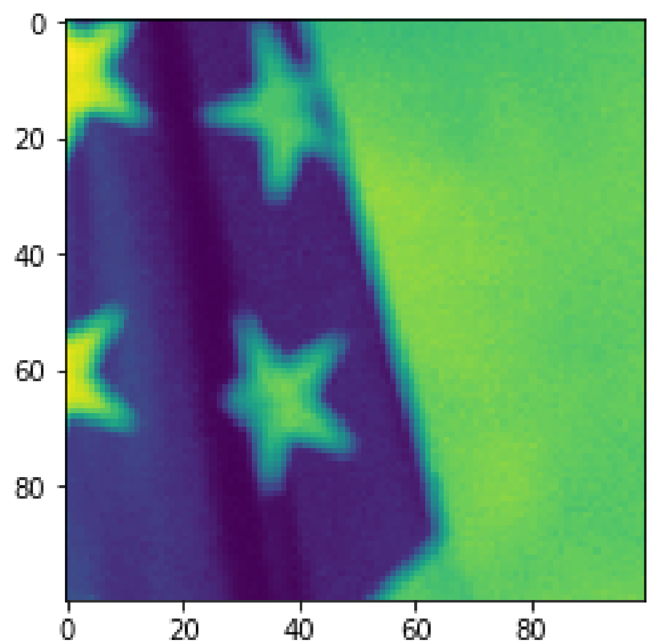


3.4.3 Subselect part of the image to show using slicing

```
In [ ]: image_data[ :100, :100, :]
```

```
In [110]: plt.imshow( image_data[ :100, :100, 0] )
```

```
Out[110]: <matplotlib.image.AxesImage at 0x12688de10>
```

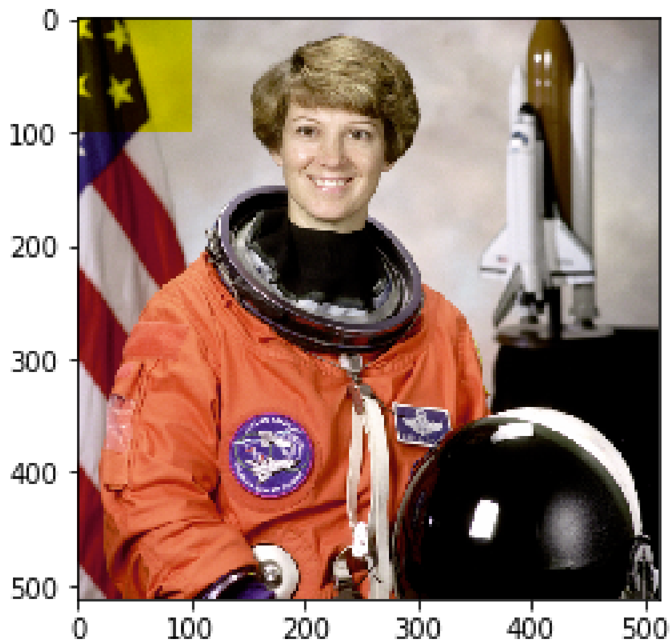


3.4.4 Zero out part of the blue channel

```
In [111]: image_data[ :100, :100, 2] = 0
```

```
In [112]: plt.imshow( image_data )
```

```
Out[112]: <matplotlib.image.AxesImage at 0x1268ed128>
```



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

```
In [113]: import pandas as pd
```

```
In [116]: titanic_data_url  = "http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic
```

```
In [117]: titanic = pd.read_excel( titanic_data_url )
```

```
In [118]: titanic.head()
```

```
Out[118]:    pclass  survived                                             name     sex  \
          0       1         1                    Allen, Miss. Elisabeth Walton  female
          1       1         1                   Allison, Master. Hudson Trevor    male
          2       1         0                     Allison, Miss. Helen Loraine  female
          3       1         0             Allison, Mr. Hudson Joshua Creighton    male
          4       1         0  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  female

                 age  sibsp  parch  ticket      fare    cabin embarked boat   body  \
          0  29.0000      0      0   24160  211.3375       B5        S    2    NaN
          1   0.9167      1      2  113781  151.5500  C22 C26        S   11    NaN
          2   2.0000      1      2  113781  151.5500  C22 C26        S  NaN    NaN
          3  30.0000      1      2  113781  151.5500  C22 C26        S  NaN  135.0
          4  25.0000      1      2  113781  151.5500  C22 C26        S  NaN    NaN

                                  home.dest
          0                    St Louis, MO
          1  Montreal, PQ / Chesterville, ON
          2  Montreal, PQ / Chesterville, ON
          3  Montreal, PQ / Chesterville, ON
          4  Montreal, PQ / Chesterville, ON
```

```
In [119]: len( titanic )
```

```
Out[119]: 1309
```

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.

```
In [120]: pd.set_option( 'display.max_rows', None )
```

4 If we have time: Breast Cancer Dataset exploration

- dataset information here
- dataset originally published here: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Di

```
In [ ]: from sklearn.datasets import load_breast_cancer
```

```
In [ ]: package = load_breast_cancer()
```

```
In [ ]: package.keys()
```

```
In [ ]: data = package['data']
```

```
In [ ]: data.shape
```

```
In [ ]: print(package['DESCR'])
```

```
In [ ]: package['feature_names']
```