Python Resources

The purpose of this document is to direct you to resources that you may find useful if you decide to do a deeper dive into Python. This course is not meant to be an introduction to programming, nor an introduction to Python, but if you find yourself interested in exploring Python further, or feel as if this is a useful skill, this document aims to direct you to resources that you may find useful. If you have a background in Python or programming, a style guides are included below to show how Python may differ from other programming languages or give you a launching point for diving deeper into more advanced packages. This course does not endorse the use or non-use of any particular resource, but the author has found these resources useful in their exploration of programming and Python in particular

The Python Documentation

Any reference that does not begin with the Python documentation would not be complete. The authors of the language, as well as the community that supports it, have developed a great set of tutorials, documentation, and references around Python. When in doubt, this is often the first place that you should look if you run into a scary error or would like to learn more about a specific function. The documentation can be found here: Python Documentation (https://docs.python.org/3/)

Python Programming Introductions

Below are resources to help you along your way in learning Python. While it is great to consume material, in programming there is no substitute for actually writing code. For every hour that you spend learning, you should spend about twice that amount of time writing code for cool problems or working out examples. Coding is best learned through actually coding!

- Coursera (https://www.coursera.org/courses?query=python) has several offerings for Python that you can take in addition to this course. These courses will go into depth into Python programming and how to use it in an applied setting
- Code Academy (https://www.codecademy.com/learn/learn-python) is another resources that is great for learning Python (and other programming languages). While not as focused as Cousera, this is a quick way to get up-and-running with Python
- YouTube is another great resource for online learning and there are several "courses" for learning Python. We recommend trying several sets of videos to see which you like best and using multiple video series to learn since each will present the material in a slightly different way
- There are tens of books on programming in Python that are great if you prefer to read. More so than the other resources, be sure to code what you learn. It is easy to read about coding, but you really learn to code by coding!
- If you have a background in coding, the authors have found the tutorial at Tutorials Point (https://www.tutorialspoint.com/python/index.htm) to be useful in getting started with Python. This tutorial assumes that you have some background in coding in another language

Cheatsheets and References

There are a variety of one-pagers and cheat-sheets available for Python that summarize the language in a few simple pages. These resources tend to be more aimed at someone who knows the language, or has experience in the language, but would like a refresher course in how the language works.

- Cheatsheet for Numpy (https://www.datacamp.com/community/blog/python-numpy-cheat-sheet#gs.AK5ZBgE)
- Cheatsheet for Datawrangling (https://www.datacamp.com/community/blog/pandas-cheat-sheet-python#gs.HPFoRIc)
- Cheatsheet for Pandas (https://www.datacamp.com/community/blog/python-pandas-cheat-sheet#gs.oundfxM)
- Cheatsheet for SciPy (https://www.datacamp.com/community/blog/python-scipy-cheat-sheet#gs.JDSg3OI)
- Cheatsheet for Matplotlib (https://www.datacamp.com/community/blog/python-matplotlib-cheat-sheet#gs.uEKySpY)

Python Style Guides

As you learn to code, you will find that you will begin to develop your own style. Sometimes this is good. Most times, this can be detrimental to your code readability and, worse, can hinder you from finding bugs in your own code in extreme cases.

It is best to learn good coding habits from the beginning and the Google Style Guide (https://github.com/google/styleguide/blob/gh-pages/pyguide.md) is a great place to start. We will mention some of these best practices here.

Consistent Indenting

Python will generally 'yell' at you if your indenting is incorrect. It is good to use an editor that takes care of this for you. In general, four spaces are preferred for indenting and you should not mix tabs and spaces.

In [1]:
```python
# Good Indenting - four spaces are standard but consistiency is key
result = []
for x in range(10):
    for y in range(5):
        if x * y > 10:
            result.append((x, y))
print (result)

# Bad indenting
result = []
for x in range(10):
  for y in range(5):
     if x * y > 10:
               result.append((x, y))
print (result)
```

```
[(3, 4), (4, 3), (4, 4), (5, 3), (5, 4), (6, 2), (6, 3), (6, 4), (7, 2), (7, 3), (7, 4), (8, 2), (8, 3), (8,
4), (9, 2), (9, 3), (9, 4)]
[(3, 4), (4, 3), (4, 4), (5, 3), (5, 4), (6, 2), (6, 3), (6, 4), (7, 2), (7, 3), (7, 4), (8, 2), (8, 3), (8,
4), (9, 2), (9, 3), (9, 4)]
```

Commenting

Comments seem weird when you first begin programming - why would I include 'code' that doesn't run? Comments are probably some of the most important aspects of code. They help other read code that is difficult for them to understand, and they, more importantly, are helpful for yourself if you look at the code in a few weeks and need clarity on why you did something. Always comment and comment well.

In [2]:
```python
################################################################################
#                                                                              #
#                               Good Commenting                                #
#                                                                              #
################################################################################

################################ Bad Commenting ################################

# My loop
for x in range(10):
    print (x)

############################## Better Commenting ###############################

# Looping from zero to ten
for x in range(10):
    print (x)

############################# Preferred Commenting #############################

# Print out the numbers from zero to ten
for x in range(10):
    print (x)
```

```
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
```

In [0]:
```
###############################################################################
#                                                                             #
#                         Mixing Commenting Strategies                        #
#                                                                             #
###############################################################################

# Try not to mix commenting styles in the same blocks - just be consistient

########### Bad - mixing doc-strings commenting and line commenting ############

''' Printing one to five, a six, and then six to nine'''
for x in range(10):
    # If x > 5, then print the value
    if x > 5:
        print (x)
    else:
        print (x + 1)

#################### Good - no mixing of comment types #######################

# Printing one to five, a six, and then six to nine
for x in range(10):
    # If x > 5, then print the value
    if x > 5:
        print (x)
    else:
        print (x + 1)
```

Line Length

Try to avoid excessively long lines. Standard practice is to keep lines to no longer than 80 characters. While this is not a hard rule, it is a good practice to follow for readability

In [0]:
```
######################### Bad - This code is too long ##########################

my_random_array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

############ Good - this code is wrapped to avoid excessive length #############

my_random_array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9,
                   10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8,
                   9, 10]
```

White Space

Utilizing Whitespace is a great way to improve the way that your code looks. In general the following can be helpful to improve the look of your code

- Try to space out your code and introduce whitespace to improve readability
- Use spacing to separate function arguments
- Do not over-do spacing. Too many spaces between code blocks makes it difficult to organize code well

In [0]:
```
################ Bad - this code has bad whitespace management #################

my_player = player()
player_attributes = get_player_attributes(my_player,height,weight,   birthday)


player_attributes[0]*=12 # convert from feet to inches




player.shoot_ball()


########################## Good whitespace management ##########################

my_player = player()
player_attributes = get_player_attributes(my_player, height, weight, birthday)

# convert from feet to inches
player_attributes[0] *= 12

player.shoot_ball()
```

The tip of the iceberg

Take a look at code out in the wild if you are really curious. How are they coding specific things? How do they manage spacing in loops? How do they manage the whitespace in argument list?

You will learn to code by coding, and you will develop your own style but starting out with good habits ensures that your code is easy to read by others and, most importantly, yourself. Good luck!