Python cheatsheet for Duke SwC bootcamp 16-17 June 2014

Lesson 1 - Language building blocks

IPython notebook

Start the IPython notebook by typing in the shell `bash ipython notebook`

To get help on how to use an object, type `python object?  object??  help(object)`

Using IPython as a calculator

To evaluate a cell use * **Alt-Enter**: run cell, insert below * **Shift-Enter**: run cell, select below * **Ctrl-Enter**: run cell

Note: comments in python are preceded by `#`. They can be placed anywhere on a line.

Python types

- int: 3, 7, 11
- float: 3.14, np.pi:
- bool: True, False, 3 <= 4
- string: 'hello', "goodbye", ""“hello again”“"
- list: [1,2,3]
- dictionary: {'a': 123, 'b': 456}
- tuple: a, b, (1,2,3)
- set: set([1,2,2,3])

Useful built-in functions

- print
- range(start, stop, step)
- type conversions: int(), float(), set(), list()

Indexing and slice notation

```
xs = ['a', 'b', 'c', 'd'
xs[0] # 'a'
xs[:2] # ['a', 'b']
xs[2:] # ['c', 'd']
xs[::2] # ['a', 'c']
xs[::-1] # ['d', 'c', 'b', 'a']
```

String methods

```
upper(), lower(), capitalize(), strip(), split(), join(), translate(), find(), count(), ljust()
```

The `+` operator concatenates two strings to form a bigger string.

List methods

```
append(), extend(), insert(),  remove(), sort(), reverse()
```

The + operator concatenates two lists to form a bigger list.

Dictionary methods

```
keys(), values(), items(), get()
```

Looping

```
for variable in sequence:
    do_something_with_variable

for index, variable in enumerate(sequence):
    do_something_with_index_and_variable
```

Control flow

```
if condition:
    do_something
else:
    do_something_else
```

List comprehension

```
[x for x in sequence if condition(x)]
```

Lesson 2 - Functions

All functions have the following structure:

```
def function_name(parameter1, parameter2, ...):
    """Optional doc string describing function."""
    body_of_function
    return value
```

and are *called* by using the function_name and its parameters `python return_value = function_name(parameter1, parameter2)`

Function parameters and arguments

For the truly pedantic, a variable used in the function call list is a "parameter". The same variable in the body of the function is called an"argument". In practice, the terms parameter and argument are used interchangeably.

```
def func(a, b, c=42): # a, b and c are parameters, and c has a default value
   do_something

# calling the function
func(x, y, z) # x, y and z are the arguments provided to func - a=x, b=y, c=z
```

Return values

```python
def func(a, b):
    return a+b

# calling the function
c = func(3, 4) # c is now 7
```

Function scope

```python
x = 3
def f():
    print x # first print statement (inside function)
    x = 4
    print x # second print statement (inside function)

# calling function
f()
print x # third print statement (outside function)
```

will have the output `python 3 # from first print statement (inside function) 4 # from second print statement (inside function) 3 # from third print statement (outside function)`

Higher order functions

Once defined, functions can be treated like any other value. In particular, functions can be values in a dictionary, serve as arguments to other functions, and functions can even return other functions.. Higher order functions are functions that take other functions as arguments and/or return functions. A classic example is `map` which applies a function to a sequence.

```python
def square(x):
    return x*x

map(square, [1, 2, 3]) # [1, 4, 9]]
```

Map and the list comprehension `[square(i) for i in [1, 2, 3]]` are essentially equivalent, and it is mostly a matter of personal preference which syntax you prefer.

Lesson 3 - Inputs, outputs and modules

Parsing text files line by line

```python
with open(<filename>, 'rU') as f:
    for line in f:
        do_something_with_line
```

Parsing CSV file row by row

```python
import csv
```

```
with open(<filename>) as f:
    reader = csv.DictReader(f)
    for row in reader:
        do_something_with_row
```

Using Pandas

```
import pandas as pd

df1 = pd.read_csv(<filename>)
df2 = pd.read_csv(<URL>) # pandas will also read remote files if a URL is given as the argument
df3 = pd.read_excel(<filename>)
```

File output

```
with open(<filename>, 'wU') as f:
    for line in lines:
        f.write(line)
```

File output with csv module

```
with open(<filename>, 'wU') as outfile:
    writer = csv.writer(outfile)

    for row in rows:
        writer.writerow(row)
```

File output with pandas

```
df1.to_csv(<filename>)
df2.to_excel(<filename>)
```

Importing moules

```
import numpy
import pandas as pd
from csv import reader, writer
from itertools import *
```

Installing modules

Do this in the shell. `bash conda search <package> # to see if a package is provided by the anaconda distribution conda install <package> # if package is in anaconda distribution pip install -U <package> # otherwise - installs from PyPI, the Python Package Index (-U means upgrade to latest version if already installed)`

BioPython

```
from Bio import Entrez, SeqIO

Entrez.search(), Entrez.read(), Entrez.efectch()
SeqIO.read(), SeqIO.write()
reverse_complement(), translate()
```

Matplotlib

```
import matplotlib.pyplot as plt

plt.figure(), plt.pie(), plt.axis(), plt.subplot(), plt.barh(), plt.xticsk(), plt.yticks(), plt.title()
```