



# Data Science Cheat Sheet

## Python Basics

### BASICS, PRINTING AND GETTING HELP

**x = 3** - Assign 3 to the variable **x**  
**print(x)** - Print the value of **x**  
**type(x)** - Return the type of the variable **x** (in this case, **int** for integer)

**help(x)** - Show documentation for the **str** data type  
**help(print)** - Show documentation for the **print()** function

### READING FILES

```
f = open("my_file.txt", "r")
file_as_string = f.read()
```

- Open the file **my\_file.txt** and assign its contents to **s**

```
import csv
f = open("my_dataset.csv", "r")
csvreader = csv.reader(f)
csv_as_list = list(csvreader)
```

- Open the CSV file **my\_dataset.csv** and assign its data to the list of lists **csv\_as\_list**

### STRINGS

**s = "hello"** - Assign the string **"hello"** to the variable **s**

```
s = """She said,
there's a good idea.
"""
```

- Assign a multi-line string to the variable **s**. Also used to create strings that contain both **"** and **'** characters

**len(s)** - Return the number of characters in **s**  
**s.startswith("hel")** - Test whether **s** starts with the substring **"hel"**

**s.endswith("lo")** - Test whether **s** ends with the substring **"lo"**

**"{} plus {} is {}".format(3,1,4)** - Return the string with the values **3**, **1**, and **4** inserted

**s.replace("e","z")** - Return a new string based on **s** with all occurrences of **"e"** replaced with **"z"**

**s.split(" ")** - Split the string **s** into a list of strings, separating on the character **" "** and return that list

### NUMERIC TYPES AND

### MATHEMATICAL OPERATIONS

**i = int("5")** - Convert the string **"5"** to the integer **5** and assign the result to **i**

**f = float("2.5")** - Convert the string **"2.5"** to the float value **2.5** and assign the result to **f**

**5 + 5** - Addition

**5 - 5** - Subtraction

**10 / 2** - Division

**5 \* 2** - Multiplication

**3 \*\* 2** - Raise **3** to the power of **2** (or  $3^2$ )

**27 \*\* (1/3)** - The 3rd root of **27** (or  $\sqrt[3]{27}$ )

**x += 1** - Assign the value of **x + 1** to **x**

**x -= 1** - Assign the value of **x - 1** to **x**

### LISTS

**l = [100,21,88,3]** - Assign a list containing the integers **100**, **21**, **88**, and **3** to the variable **l**

**l = list()** - Create an empty list and assign the result to **l**

**l[0]** - Return the first value in the list **l**

**l[-1]** - Return the last value in the list **l**

**l[1:3]** - Return a slice (list) containing the second and third values of **l**

**len(l)** - Return the number of elements in **l**

**sum(l)** - Return the sum of the values of **l**

**min(l)** - Return the minimum value from **l**

**max(l)** - Return the maximum value from **l**

**l.append(16)** - Append the value **16** to the end of **l**

**l.sort()** - Sort the items in **l** in ascending order

**" ".join(["A","B","C","D"])** - Converts the list **["A","B","C","D"]** into the string **"A B C D"**

### DICTIONARIES

**d = {"CA":"Canada","GB":"Great Britain","IN":"India"}** - Create a dictionary with keys of **"CA"**, **"GB"**, and **"IN"** and corresponding values of **"Canada"**, **"Great Britain"**, and **"India"**

**d["GB"]** - Return the value from the dictionary **d** that has the key **"GB"**

**d.get("AU","Sorry")** - Return the value from the dictionary **d** that has the key **"AU"**, or the string **"Sorry"** if the key **"AU"** is not found in **d**

**d.keys()** - Return a list of the keys from **d**

**d.values()** - Return a list of the values from **d**

**d.items()** - Return a list of (**key**, **value**) pairs from **d**

### MODULES AND FUNCTIONS

*The body of a function is defined through indentation.*

**import random** - Import the module **random**

**from math import sqrt** - Import the function **sqrt** from the module **math**

```
def calculate(addition_one,addition_two,
exponent=1,factor=1):
    result = (value_one + value_two) ** exponent * factor
    return result
```

- Define a new function **calculate** with two required and two optional named arguments which calculates and returns a result.

**addition(3,5,factor=10)** - Run the **addition** function with the values **3** and **5** and the named argument **10**

### BOOLEAN COMPARISONS

**x == 5** - Test whether **x** is equal to **5**

**x != 5** - Test whether **x** is not equal to **5**

**x > 5** - Test whether **x** is greater than **5**

**x < 5** - Test whether **x** is less than **5**

**x >= 5** - Test whether **x** is greater than or equal to **5**

**x <= 5** - Test whether **x** is less than or equal to **5**

**x == 5 or name == "alfred"** - Test whether **x** is equal to **5** or **name** is equal to **"alfred"**

**x == 5 and name == "alfred"** - Test whether **x** is equal to **5** and **name** is equal to **"alfred"**

**5 in l** - Checks whether the value **5** exists in the list **l**

**"GB" in d** - Checks whether the value **"GB"** exists in the keys for **d**

### IF STATEMENTS AND LOOPS

*The body of if statements and loops are defined through indentation.*

```
if x > 5:
    print("{} is greater than five".format(x))
elif x < 0:
    print("{} is negative".format(x))
else:
    print("{} is between zero and five".format(x))
```

- Test the value of the variable **x** and run the code body based on the value

**for value in l:**

```
    print(value)
```

- Iterate over each value in **l**, running the code in the body of the loop with each iteration

**while x < 10:**

```
    x += 1
```

- Run the code in the body of the loop until the value of **x** is no longer less than **10**



# Data Science Cheat Sheet

## Python - Intermediate

### KEY BASICS, PRINTING AND GETTING HELP

*This cheat sheet assumes you are familiar with the content of our Python Basics Cheat Sheet*

**s** - A Python string variable  
**i** - A Python integer variable  
**f** - A Python float variable

**l** - A Python list variable  
**d** - A Python dictionary variable

### LISTS

**l.pop(3)** - Returns the fourth item from **l** and deletes it from the list  
**l.remove(x)** - Removes the first item in **l** that is equal to **x**  
**l.reverse()** - Reverses the order of the items in **l**  
**l[1::2]** - Returns every second item from **l**, commencing from the 1st item  
**l[-5:]** - Returns the last 5 items from **l** specific axis

### STRINGS

**s.lower()** - Returns a lowercase version of **s**  
**s.title()** - Returns **s** with the first letter of every word capitalized  
**"23".zfill(4)** - Returns **"0023"** by left-filling the string with **0**'s to make it's length **4**.  
**s.splitlines()** - Returns a list by splitting the string on any newline characters.  
*Python strings share some common methods with lists*  
**s[:5]** - Returns the first 5 characters of **s**  
**"fri" + "end"** - Returns **"friend"**  
**"end" in s** - Returns **True** if the substring **"end"** is found in **s**

### RANGE

*Range objects are useful for creating sequences of integers for looping.*  
**range(5)** - Returns a sequence from **0** to **4**  
**range(2000,2018)** - Returns a sequence from **2000** to **2017**  
**range(0,11,2)** - Returns a sequence from **0** to **10**, with each item incrementing by **2**  
**range(0,-10,-1)** - Returns a sequence from **0** to **-9**  
**list(range(5))** - Returns a list from **0** to **4**

### DICTIONARIES

**max(d, key=d.get)** - Return the key that corresponds to the largest value in **d**  
**min(d, key=d.get)** - Return the key that corresponds to the smallest value in **d**

### SETS

**my\_set = set(l)** - Return a **set** object containing the unique values from **l**

**len(my\_set)** - Returns the number of objects in **my\_set** (or, the number of unique values from **l**)  
**a in my\_set** - Returns **True** if the value **a** exists in **my\_set**

### REGULAR EXPRESSIONS

**import re** - Import the Regular Expressions module  
**re.search("abc",s)** - Returns a **match** object if the regex **"abc"** is found in **s**, otherwise **None**  
**re.sub("abc","xyz",s)** - Returns a string where all instances matching regex **"abc"** are replaced by **"xyz"**

### LIST COMPREHENSION

*A one-line expression of a for loop*  
**[i \*\* 2 for i in range(10)]** - Returns a list of the squares of values from **0** to **9**  
**[s.lower() for s in l\_strings]** - Returns the list **l\_strings**, with each item having had the **.lower()** method applied  
**[i for i in l\_floats if i < 0.5]** - Returns the items from **l\_floats** that are less than **0.5**

### FUNCTIONS FOR LOOPING

**for i, value in enumerate(l):**  
    **print("The value of item {} is {}".format(i,value))**  
- Iterate over the list **l**, printing the index location of each item and its value  
**for one, two in zip(l\_one,l\_two):**  
    **print("one: {}, two: {}".format(one,two))**  
- Iterate over two lists, **l\_one** and **l\_two** and print each value  
**while x < 10:**  
    **x += 1**  
- Run the code in the body of the loop until the value of **x** is no longer less than **10**

### DATETIME

**import datetime as dt** - Import the **datetime** module  
**now = dt.datetime.now()** - Assign **datetime** object representing the current time to **now**  
**wks4 = dt.datetime.timedelta(weeks=4)**  
- Assign a **timedelta** object representing a timespan of 4 weeks to **wks4**

**now - wks4** - Return a **datetime** object representing the time 4 weeks prior to **now**  
**newyear\_2020 = dt.datetime(year=2020, month=12, day=31)** - Assign a **datetime** object representing December 25, 2020 to **newyear\_2020**  
**newyear\_2020.strftime("%A, %b %d, %Y")**  
- Returns **"Thursday, Dec 31, 2020"**  
**dt.datetime.strptime('Dec 31, 2020', "%b %d, %Y")** - Return a **datetime** object representing December 31, 2020

### RANDOM

**import random** - Import the **random** module  
**random.random()** - Returns a random float between **0.0** and **1.0**  
**random.randint(0,10)** - Returns a random integer between **0** and **10**  
**random.choice(l)** - Returns a random item from the list **l**

### COUNTER

**from collections import Counter** - Import the **Counter** class  
**c = Counter(l)** - Assign a **Counter** (dict-like) object with the counts of each unique item from **l**, to **c**  
**c.most\_common(3)** - Return the 3 most common items from **l**

### TRY/EXCEPT

Catch and deal with Errors  
**l\_ints = [1, 2, 3, "", 5]** - Assign a list of integers with one missing value to **l\_ints**  
**l\_floats = []**  
**for i in l\_ints:**  
    **try:**  
        **l\_floats.append(float(i))**  
    **except:**  
        **l\_floats.append(i)**  
- Convert each value of **l\_ints** to a float, catching and handling **ValueError: could not convert string to float:** where values are missing.