



# Data Science Cheat Sheet

## Python Basics

### BASICS, PRINTING AND GETTING HELP

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

### 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<sup>2</sup>)

**27 \*\* (1/3)** - The 3rd root of 27 (or <sup>3</sup>√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.



# Data Science Cheat Sheet

## Numpy

### KEY

We'll use shorthand in this cheat sheet

**arr** - A numpy Array object

### IMPORTS

Import these to start

`import numpy as np`

### IMPORTING/EXPORTING

`np.loadtxt('file.txt')` - From a text file

`np.genfromtxt('file.csv', delimiter=',')`  
- From a CSV file

`np.savetxt('file.txt', arr, delimiter=' ')`  
- Writes to a text file

`np.savetxt('file.csv', arr, delimiter=',')`  
- Writes to a CSV file

### CREATING ARRAYS

`np.array([1,2,3])` - One dimensional array

`np.array([(1,2,3), (4,5,6)])` - Two dimensional array

`np.zeros(3)` - 1D array of length 3 all values 0

`np.ones((3,4))` - 3x4 array with all values 1

`np.eye(5)` - 5x5 array of 0 with 1 on diagonal (Identity matrix)

`np.linspace(0,100,6)` - Array of 6 evenly divided values from 0 to 100

`np.arange(0,10,3)` - Array of values from 0 to less than 10 with step 3 (eg [0,3,6,9])

`np.full((2,3),8)` - 2x3 array with all values 8

`np.random.rand(4,5)` - 4x5 array of random floats between 0-1

`np.random.rand(6,7)*100` - 6x7 array of random floats between 0-100

`np.random.randint(5, size=(2,3))` - 2x3 array with random ints between 0-4

### INSPECTING PROPERTIES

`arr.size` - Returns number of elements in **arr**

`arr.shape` - Returns dimensions of **arr** (rows, columns)

`arr.dtype` - Returns type of elements in **arr**

`arr.astype(dtype)` - Convert **arr** elements to type **dtype**

`arr.tolist()` - Convert **arr** to a Python list

`np.info(np.eye)` - View documentation for **np.eye**

### COPYING/SORTING/RESHAPING

`np.copy(arr)` - Copies **arr** to new memory

`arr.view(dtype)` - Creates view of **arr** elements with type **dtype**

`arr.sort()` - Sorts **arr**

`arr.sort(axis=0)` - Sorts specific axis of **arr**

`two_d_arr.flatten()` - Flattens 2D array **two\_d\_arr** to 1D

`arr.T` - Transposes **arr** (rows become columns and vice versa)

`arr.reshape(3,4)` - Reshapes **arr** to 3 rows, 4 columns without changing data

`arr.resize((5,6))` - Changes **arr** shape to 5x6 and fills new values with 0

### ADDING/REMOVING ELEMENTS

`np.append(arr, values)` - Appends **values** to end of **arr**

`np.insert(arr, 2, values)` - Inserts **values** into **arr** before index 2

`np.delete(arr, 3, axis=0)` - Deletes row on index 3 of **arr**

`np.delete(arr, 4, axis=1)` - Deletes column on index 4 of **arr**

### COMBINING/SPLITTING

`np.concatenate((arr1, arr2), axis=0)` - Adds **arr2** as rows to the end of **arr1**

`np.concatenate((arr1, arr2), axis=1)` - Adds **arr2** as columns to end of **arr1**

`np.split(arr, 3)` - Splits **arr** into 3 sub-arrays

`np.hsplit(arr, 5)` - Splits **arr** horizontally on the 5th index

### INDEXING/SLICING/SUBSETTING

`arr[5]` - Returns the element at index 5

`arr[2,5]` - Returns the 2D array element on index [2][5]

`arr[1]=4` - Assigns array element on index 1 the value 4

`arr[1,3]=10` - Assigns array element on index [1][3] the value 10

`arr[0:3]` - Returns the elements at indices 0,1,2 (On a 2D array: returns rows 0,1,2)

`arr[0:3,4]` - Returns the elements on rows 0,1,2 at column 4

`arr[:2]` - Returns the elements at indices 0,1 (On a 2D array: returns rows 0,1)

`arr[:,1]` - Returns the elements at index 1 on all rows

`arr<5` - Returns an array with boolean values (**arr1<3** & **arr2>5**) - Returns an array with boolean values

`~arr` - Inverts a boolean array

`arr[arr<5]` - Returns array elements smaller than 5

### SCALAR MATH

`np.add(arr,1)` - Add 1 to each array element

`np.subtract(arr,2)` - Subtract 2 from each array element

`np.multiply(arr,3)` - Multiply each array element by 3

`np.divide(arr,4)` - Divide each array element by 4 (returns **np.nan** for division by zero)

`np.power(arr,5)` - Raise each array element to the 5th power

### VECTOR MATH

`np.add(arr1, arr2)` - Elementwise add **arr2** to **arr1**

`np.subtract(arr1, arr2)` - Elementwise subtract **arr2** from **arr1**

`np.multiply(arr1, arr2)` - Elementwise multiply **arr1** by **arr2**

`np.divide(arr1, arr2)` - Elementwise divide **arr1** by **arr2**

`np.power(arr1, arr2)` - Elementwise raise **arr1** raised to the power of **arr2**

`np.array_equal(arr1, arr2)` - Returns **True** if the arrays have the same elements and shape

`np.sqrt(arr)` - Square root of each element in the array

`np.sin(arr)` - Sine of each element in the array

`np.log(arr)` - Natural log of each element in the array

`np.abs(arr)` - Absolute value of each element in the array

`np.ceil(arr)` - Rounds up to the nearest int

`np.floor(arr)` - Rounds down to the nearest int

`np.round(arr)` - Rounds to the nearest int

### STATISTICS

`np.mean(arr, axis=0)` - Returns mean along specific axis

`arr.sum()` - Returns sum of **arr**

`arr.min()` - Returns minimum value of **arr**

`arr.max(axis=0)` - Returns maximum value of specific axis

`np.var(arr)` - Returns the variance of array

`np.std(arr, axis=1)` - Returns the standard deviation of specific axis

`arr.corrcoef()` - Returns correlation coefficient of array



# Data Science Cheat Sheet

## Python Regular Expressions

### SPECIAL CHARACTERS

- ^** | Matches the expression to its right at the start of a string. It matches every such instance before each **\n** in the string.
- \$** | Matches the expression to its left at the end of a string. It matches every such instance before each **\n** in the string.
- .** | Matches any character except line terminators like **\n**.
- \** | Escapes special characters or denotes character classes.
- A|B** | Matches expression **A** or **B**. If **A** is matched first, **B** is left untried.
- +** | Greedily matches the expression to its left 1 or more times.
- \*** | Greedily matches the expression to its left 0 or more times.
- ?** | Greedily matches the expression to its left 0 or 1 times. But if **?** is added to qualifiers (**+**, **\***, and **?** itself) it will perform matches in a non-greedy manner.
- {m}** | Matches the expression to its left **m** times, and not less.
- {m,n}** | Matches the expression to its left **m** to **n** times, and not less.
- {m,n}?** | Matches the expression to its left **m** times, and ignores **n**. See **?** above.

### CHARACTER CLASSES

#### [A.K.A. SPECIAL SEQUENCES]

- \w** | Matches alphanumeric characters, which means **a-z**, **A-Z**, and **0-9**. It also matches the underscore, **\_**.
- \d** | Matches digits, which means **0-9**.
- \D** | Matches any non-digits.
- \s** | Matches whitespace characters, which include the **\t**, **\n**, **\r**, and space characters.
- \S** | Matches non-whitespace characters.
- \b** | Matches the boundary (or empty string) at the start and end of a word, that is, between **\w** and **\W**.
- \B** | Matches where **\b** does not, that is, the boundary of **\w** characters.

**\A** | Matches the expression to its right at the absolute start of a string whether in single or multi-line mode.

**\Z** | Matches the expression to its left at the absolute end of a string whether in single or multi-line mode.

### SETS

- [ ]** | Contains a set of characters to match.
- [amk]** | Matches either **a**, **m**, or **k**. It does not match **amk**.
- [a-z]** | Matches any alphabet from **a** to **z**.
- [a\ -z]** | Matches **a**, **-**, or **z**. It matches **-** because **\** escapes it.
- [a -]** | Matches **a** or **-**, because **-** is not being used to indicate a series of characters.
- [-a]** | As above, matches **a** or **-**.
- [a-z0-9]** | Matches characters from **a** to **z** and also from **0** to **9**.
- [+\*?]** | Special characters become literal inside a set, so this matches **(**, **+**, **\***, and **)**.
- ^[ab5]** | Adding **^** excludes any character in the set. Here, it matches characters that are not **a**, **b**, or **5**.

### GROUPS

- ( )** | Matches the expression inside the parentheses and groups it.
- (?)** | Inside parentheses like this, **?** acts as an extension notation. Its meaning depends on the character immediately to its right.
- (?PAB)** | Matches the expression **AB**, and it can be accessed with the group name.
- (?aiLmsux)** | Here, **a**, **i**, **L**, **m**, **s**, **u**, and **x** are flags:
  - a** — Matches ASCII only
  - i** — Ignore case
  - L** — Locale dependent
  - m** — Multi-line
  - s** — Matches all
  - u** — Matches unicode
  - x** — Verbose

**(?:A)** | Matches the expression as represented by **A**, but unlike **(?PAB)**, it cannot be retrieved afterwards.

**(?#...)** | A comment. Contents are for us to read, not for matching.

**A(?:=B)** | Lookahead assertion. This matches the expression **A** only if it is followed by **B**.

**A(?:!B)** | Negative lookahead assertion. This matches the expression **A** only if it is not followed by **B**.

**(?<=B)A** | Positive lookbehind assertion.

This matches the expression **A** only if **B** is immediately to its left. This can only matched fixed length expressions.

**(?<!B)A** | Negative lookbehind assertion.

This matches the expression **A** only if **B** is not immediately to its left. This can only matched fixed length expressions.

**(?P=name)** | Matches the expression matched by an earlier group named "name".

**(...)\1** | The number **1** corresponds to the first group to be matched. If we want to match more instances of the same expression, simply use its number instead of writing out the whole expression again. We can use from **1** up to **99** such groups and their corresponding numbers.

### POPULAR PYTHON RE MODULE FUNCTIONS

- re.findall(A, B)** | Matches all instances of an expression **A** in a string **B** and returns them in a list.
- re.search(A, B)** | Matches the first instance of an expression **A** in a string **B**, and returns it as a re match object.
- re.split(A, B)** | Split a string **B** into a list using the delimiter **A**.
- re.sub(A, B, C)** | Replace **A** with **B** in the string **C**.