PYTHON CHEAT SHEET FOR DATA SCIENCE: THE BASICS

PYTHON DATA TYPES

- String: Series of characters or data stored as text ('ABC')
- Integer: A whole number (15)
- Float: A decimal number (1.5)
- Boolean: Discrete value true or false (True)
- Dictionary: Changeable collection of key-value pairs {'A':15}
- Tuple: Unchangeable collection of objects (1,5)
- List: Changeable collection of objects [1,5]
- Set: Unordered collection of unique objects {1,5}

STRING OPERATIONS

- `my_string.upper()` : Returns the string with all uppercase letters
- `len(my_string)` : Returns the length of a string
- `my_string.find('l')` : Returns the index of the first instance of the string inside the subject string, otherwise -1
- `my_string.replace('H', 'C')` : Replaces any instance of the first string with the second in `my_string`

LIST OPERATIONS

- `len(my_collection)` : Returns the length of a list
- `my_collection.extend(['More', 'Items'])` : Add multiple items to a list
- `my_collection.append('Single')` : Add a single item to a list
- `del(my_collection[2])` : Delete the object of a list at a specified index
- `clone = my_collection[:]` : Clone a list
- `my_collection_3 = my_collection + my_collection_2` : Concatenate two lists
- `sum(number_collection)` : Calculate the sum of a list of ints or floats
- `item in my_collection` : Check if an item is in a list, returns Boolean
- `item not in my_collection` : Check if an item is not in a list, returns Boolean

DICTIONARY OPERATIONS

- `my_dictionary['banana']` : Access value using key
- `my_dictionary.keys()` : Get all keys in a dictionary as a list
- `my_dictionary.values()` : Get all values in a dictionary as a list

SET OPERATIONS

- `my_set = set([1, 1, 2, 3])` : Convert a list to a set
- `a.add(4)` : Add an item to the set
- `a.remove('Bye')` : Remove an item from a set
- `a.difference(b)` : Returns set `a` minus `b`
- `a.intersection(b)` : Returns the intersection of set `a` and `b`
- `a.union(b)` : Returns the union of set `a` and `b`
- `a.issubset(b)` : Returns True if `a` is a subset of `b`, false otherwise
- `a.issuperset(b)` : Returns True if `a` is a superset of `b`, false otherwise

INDEXING AND SLICING

- Indexing: Accessing data from a string, list, or tuple using an element number
- `my_string[element_number]`
- `my_collection[element_number]`
- `my_tup[element_number]`
- Slicing: Accessing a subset of data from a string, list, or tuple using element numbers from start to stop -1
- `my_string[start:stop]`
- `my_collection[start:stop]`
- `my_tup[start:stop]`

COMPARISON OPERATORS

- Equal: `a == b`
- Less Than: `a < b`
- Greater Than: `a > b`
- Greater Than or Equal: `a >= b`
- Less Than or Equal: `a <= b`
- Not Equal: `a != b`

PYTHON OPERATORS

- `+`: Addition
- `-`: Subtraction
- `*`: Multiplication
- `/`: Division
- `//`: Integer Division (Result rounded to the nearest integer)

CONDITIONAL OPERATORS

- And: Returns true if both statements `a` and `b` are true, otherwise false
- Or: Returns true if either statement `a` or `b` is true, otherwise false
- Not: Returns the opposite of the statement
- `not a`

LOOPS

For Loops:

- `for x in range(x):` Executes loop `x` number of times
- `for x in iterable:` Executes loop for each object in an iterable like a string, tuple, list, or set

While Loops:

- `while statement:` Executes the loop while statement is true

```
# For loop example:
Print numbers from 1 to 5
for i in range(1, 6):
    print(i)
# While loop example:
Print numbers from 1 to 5
n = 1
while n <= 5:
    print(n) n += 1
```

CONDITIONAL STATEMENTS

- `if statement_1:` Execute if `statement_1` is true
- `elif statement_2:` Execute if `statement_1` is false and `statement_2` is true
- `else:` Execute if all previous statements are false

```
if num > 0:
    print("The number is positive." )
elif num < 0:
    print("The number is negative." )
else:
    print("The number is zero.“)
```

TRY/EXCEPT

- `try:` Code to try to execute
- `except a:` Code to execute if there is an error of type `a`
- `except b:` Code to execute if there is an error of type `b`
- `except:` Code to execute if there is any exception that has not been handled
- `else:` Code to execute if there is no exception

ERROR TYPES

- `IndexError:` When an index is out of range
- `NameError:` When a variable name is not found
- `SyntaxError:` When there is an error with how the code is written
- `ZeroDivisionError:` When your code tries to divide by zero

RANGE

- Produce an iterable sequence from 0 to stop-1: `range(stop)`
- Produce an iterable sequence from start to stop-1 incrementing by step: `range(start, stop, step)`

WEBSCRAPING

- `from bs4 import BeautifulSoup`: Import BeautifulSoup
- `soup = BeautifulSoup(html, 'html5lib')`: Parse HTML stored as a string
- `soup.prettify()`: Returns formatted HTML
- `soup.find(tag)`: Find the first instance of an HTML tag
- `soup.find_all(tag)`: Find all instances of an HTML tag

REQUESTS

- `import requests`: Import the requests library
- `response = requests.get(url, parameters)`: Send a get request to the URL with optional parameters
- `response.url`: Get the URL of the response
- `response.status_code`: Get the status code of the response
- `response.request.headers`: Get the headers of the request
- `response.request.body`: Get the body of the request
- `response.headers`: Get the headers of the response
- `response.text`: Get the content of the response in text
- `response.json()`: Get the content of the response in JSON
- `requests.post(url, parameters)`: Send a post request to the URL with optional parameters

FUNCTIONS

- `def function_name(optional_parameter_1, optional_parameter_2):`: Create a function
- `return optional_output`: Code to execute
- `output = function_name(parameter_1, parameter_2)`: Calling a function

```
def factorial(n):
    if n == 0:
        return 1
    else: return n * factorial(n-1) # Example usage result = factorial(5)
print("Factorial of 5 is:", result)
```

WORKING WITH FILES

Reading a File:

- `file = open(file_name, "r")`: Opens a file in read mode
- `file.name`: Returns the file name
- `file.mode`: Returns the mode the file was opened in
- `file.read()`: Reads the contents of a file
- `file.read(characters)`: Reads a certain number of characters of a file
- `file.readline()`: Read a single line of a file
- `file.readlines()`: Read all the lines of a file and stores it in a list
- `file.close()`: Closes a file
- Writing to a File:
- `file = open(file_name, "w")`: Opens a file in write mode
- `file.write(content)`: Writes content to a file
- `file.append(content)`: Adds content to the end of a file

OBJECTS AND CLASSES

Creating a class:

- `class class_name:`
- `def __init__(self, optional_parameter_1, optional_parameter_2):`
- `self.attribute_1 = optional_parameter_1`
- `self.attribute_2 = optional_parameter_2`
- `def method_name(self, optional_parameter_1):`
- Code to execute
- `return optional_output`
- Create an instance of a class:
- `object = class_name(parameter_1, parameter_2)`
- Calling an object method:
- `object.method_name(parameter_3)`