PYTHON 3 CHEAT SHEET

SUPER DATASCIENCE
MAKING THE COMPLEX SIMPLE

1. VARIABLE ASSIGNMENT AND STRINGS

Strings are a series of characters, variables are used for data storage.

1.1 PRINT A STRING:

```
print ("I am becoming a pro in python!")
```

1.2 PRINT WITH A VARIABLE

```
var = "I am becoming a pro in python!"
print(var)
```

1.3 PRINT WITH A FORMAT

```
name = "Chanel"
age = 26
print("My name is {} and
      I am {} years old".format(name, age))
```

1.4 STRING CONCATENATION

```
first_name = "Barack"
last_name = "Obama"
full_name = first_name + " " + last_name
print(full_name)
```

1.5 STRING METHODS

Convert to upper case: `var.upper()`

Split a string into words: `var.split()`

2. LISTS

List is a collection of items that are ordered and changeable. List allows duplicate members and can be accessed using indexing.

2.1 CREATE A LIST

```
my_list = [2, "Mitch", 5]
```

2.2 GET AN ITEM FROM LIST USING INDEXING

First element: `my_list[0]`

Last element: `my_list[-1]`

2.3 SLICE A LIST (GET MULTIPLE ELEMENTS)

```
my_list = [2, "Mitch", 5, 7, 8, 10, 15]
```

Get 2nd to 4th element: `my_list[1:4]`

Get all elements in a list: `my_list[:]`

2.4 GET NUMBER OF ELEMENTS IN LIST

```
len(my_list)
```

2.5 APPEND ITEMS TO A LIST

```
names = []
names.append('Jennifer Anniston')
names.append('Angelina Jolie')
```

2.6 LOOP ON A LIST OF STRINGS

```
my_list = ["apple", "blueberries",
           "mango", "watermelon"]
for i in my_list:
    print(i)
```

2.7 LOOP ON A LIST OF NUMERICAL ELEMENTS

```
my_list = [1, 2, 3, 4]
output_list = []
for element in my_list:
    output_list.append(element**2)
```

2.8 LIST COMPREHENSION

```
my_list = [1, 2, 3, 4]
[element**2 for element in my_list]
```

3. DICTIONARIES

Dictionaries store elements in a key-value pair format. Dictionary elements are accessed via keys while List elements are accessed by their index

3.1 DEFINING A DICTIONARY

```
my_dict = { "brand": "Apple",
            "model": "iphone X",
            "year" : 2018}
```

3.2 ACCESSING A VALUE

```
my_dict["model"]
```

3.3 ADDING KEY VALUE PAIR

```
my_dict["color"] = "red"
```

3.4 REMOVING KEY VALUE PAIR

```
del my_dict["brand"]
```

3.5 DELETE THE ENTIRE DICTIONARY

```
del my_dict
```

4. TUPLES

A tuple is a sequence of immutable Python objects.

4.1 DEFINING A TUPLE

```
tuple_1 = ('Mitch', 'Chanel', 10, 15, 1992);
```

4.2 ACCESSING ELEMENTS IN A TUPLE

```
tuple_1 = ('Mitch', 'Chanel', 10, 15, 1992);
```

First element: `tuple_1[0]`

Tuple slicing 2nd to 3rd element: `tuple_1[1:3]`

4.3 WHAT'S NOT ALLOWED!

```
tuple_1[1] = 0
```

5. SETS

A set is an unordered collection of items. Every element is unique (no duplicates).

5.1 DEFINING A SET

```
my_set = {1,2,3,4,3,2}
```

5.2 CREATE A SET FROM A LIST

```
my_list = [1,2,3,2]
my_set = set(my_list)
```

6. ENTER VALUE FROM USER

Prompt a user for input and store it in a variable

```
name = input("Welcome to Python course,
              what's your name? ")
print("Welcome " + name + ":)")
```

7. COMPARISON OPERATORS/ IF-ELSE STATEMENTS

7.1 COMPARISON OPERATORS:

Used to compare 2 or more values and decide if the condition is True or False

Equals: `x == 5`
Not equal: `x != 5`
Greater than: `x > 5`
Greater than or equal: `x >= 5`
Less than: `x < 5`
Less than or equal: `to x <= 5`

7.2 CONDITIONAL STATEMENTS (IF-ELSE)

If statement can be followed by an optional else statement, which executes when the boolean expression is FALSE.

```
if 5 > 2:
    print('If condition is True')
else:
    print('If condition is False')
```

7.3 MUTLIPLE IF-ELSE STATEMENTS

You can use one if or else if statement inside another if or else if statement(s).

```
if 5 == 7:
    print('first statement is True')
elif 2 == 4:
    print('Second Statement is True')
else:
    print('Last Statement is True')
```

8. FOR LOOPS

8.1 DEFINE A LOOP

```
my_list = [1, 2, 3]
for i in my_list:
    print(i)
```

8.2 ENUMERATE

Enumerate allows us to loop over something and have an automatic counter

```
for i, element in enumerate(my_list):
    print(i, element)
```

8.3 BREAK A LOOP

Exit the loop when a condition is satisfied

```
my_list = ["apple", "blueberries", "mango",
           "watermelon", "apricots"]
for i in my_list:
    print(i)
    if i == "mango":
        break
```

8.4 CONTINUE STATEMENT

continue() is used to skip the current block, and return to the "for" or "while" statement.

```
for i in range(20):
    # Check if i is even
    if i % 2 == 0:
        continue # Don't do anything and return
                 again to for loop!
    print(i)
```

8.5 WHILE LOOPS

While loop can be used to execute a set of statements as long as a certain condition holds true.

```
i = 0
while i <= 10:
    print(i)
    i += 1
```

8.6 NESTED LOOPS

Nested loops are loops that exist inside the body of another loop

```
# Print the multiplication table
for x in range(1, 6):
    for y in range(1, 11):
        print ('{} * {} = {}'.format(x, y, x*y))
```

9. FUNCTIONS

A function is a block of code that can run whenever it is called. Information passed to a function is called an argument.

9.1 FUNCTION DEFINITION

```
def squared(x):
    return x**2
```

9.2 FUNCTION CALL

```
print(squared(2))
```

9.3 SET A DEFAULT PARAMETER VALUE

```
def my_function(age = 25):
    print("I am {} years old".format(age))
```

9.4 LAMBDA EXPRESSIONS

Lambda function is used to create small elegant anonymous functions, generally used with filter() and map()

```
y = lambda x:x**2
y(3)
```

9.5 MAP

The map() function takes in a function and a list. The function perform an operation on the entire list and return the results in a new list.

```
my_list = [1, 2, 3, 5]
output_list = list(map( lambda x: x**2 , my_list))
print(output_list)
```

9.6 FILTER

Perform an operation on a list based on a specific condition (after filtering!)

```
mylist_age = [5, 15, 18, 21, 22, 24, 39, 35]
above_drinking_age = list(filter(lambda x:
                                 (x >= 21), mylist_age))
print(above_drinking_age)
```

10. FILES OPERATIONS

fopen() *is the key function to handle files and it takes two parameters: filename, and mode.*

Modes for files opening:

- ***"r" - Read*** *- Default value. Opens a file for reading, error if the file does not exist*
- ***"a" - Append*** *- Opens a file for appending, creates the file if it does not exist*
- ***"w" - Write*** *- Opens a file for writing, creates the file if it does not exist*
- ***"x" - Create*** *- Creates the specified file, returns an error if the file exists*

10.1 READ FILES

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

10.2 APPEND TO FILES

```
f = open("my_file.txt", "a")
f.write("I am learning how to handle files
         in python!")
```

10.3 CREATE A NEW FILE

```
f = open("myfile_new.txt", "x")
f.write("I am learning how to handle files
         in python!")
f.close()
```