Python Crash Course - Cheatsheet

In [4]:
```
# https://ehmatthes.github.io/pcc/cheatsheets/README.html
```

Roadmaps for Developers (Frontend/Backend/DevOps)

In [ ]:
```
https://roadmap.sh/
```

Variables

In [1]:
```
a = 1        # integer
b = 1.1      # float
c = 1 + 2j   # complex number (a + bi)
d = "a"     # string
e = True     # boolean (True / False)

#print(e)
print(f"{a},{b},{c},{d},{e}")
```

```
1,1.1,(1+2j),a,True
```

Strings

In [2]:
```
# Escape sequences

#\' Single Quote(')
my_message1 = 'John\'s book'

#\" Double Quote(")
my_message2 = "I live in \"UK\""

#\\ Backslash(\)
a = "and"
b = "or"
my_message3 = (f"{a}\\{b}")

#\n New Line (ASCII Linefeed(LF))
print("Hello World")
print("\n") # will add a new line
print(my_message1)
print(my_message2)
print(my_message3)
```

```
Hello World


John's book
I live in "UK"
and\or
```

In [3]:
```
# Formatted strings
first = "John"
last = "Smith"
name = (f"{first},{last}")
print(name)

# String methods
x = "my name is john. I LIVE IN U.K"

#It'll print the string in upper case
print(x.upper())

#It'll print the string in lower case
print(x.lower())

#It'll print the string in title case
print(x.title())

print('\n')

#Strip() - method removes characters from both left
#and right based on the argument(a string specifying
#the set of characters to be removed).

#it'll remove the space before (and after).
#Check the output.
print("---------")
y = " my name is John."
print(y)
print(y.strip())
print("----------")

#Find and replace
z1 = "ny name is John."
print(z1)
z1.find("ny")
print(z1.replace("ny","my"))
```

```
John,Smith
MY NAME IS JOHN. I LIVE IN U.K
my name is john. i live in u.k
My Name Is John. I Live In U.K


---------
 my name is John.
my name is John.
----------
ny name is John.
my name is John.
```

In [4]:
```
#String indexing/slicing

message = 'Hello world'
print(message[0])
print(message[-1])
print(message[0:5])
```

```
H
d
Hello
```

Data Structures:

In [39]:
```
#https://data36.com/python-data-structures-data-science-basics/

"""
There are three major Python data structures:

# Lists

book_list = ['A Game of Thrones', 'Digital Fortress',
'Practical Statistics for Data Scientists']

# Tuples

book_tuple = ('A Game of Thrones', 'Digital Fortress',
'Practical Statistics for Data Scientists')


# Dictionaries
book_dictionary = {'George R. R. Martin': 'A Game of Thrones',
                'Dan Brown': 'Digital Fortress',
                'A. & P. Bruce': 'Practical Statistics for Data Scientis
ts'}

"""

#https://data36.com/python-data-structures-data-science-basics/
```

Out[39]:
```
"\nThere are three major Python data structures:\n\n# Lists\n\nbook_lis
t = ['A Game of Thrones', 'Digital Fortress',\n'Practical Statistics fo
r Data Scientists']\n\n# Tuples\n\nbook_tuple = ('A Game of Thrones',
'Digital Fortress', \n'Practical Statistics for Data Scientists')\n\n\n
# Dictionaries\nbook_dictionary = {'George R. R. Martin': 'A Game of Th
rones', \n                'Dan Brown': 'Digital Fortress', \n
'A. & P. Bruce': 'Practical Statistics for Data Scientists'}\n\n"
```

Type Conversion

In [6]:
```
"""Python defines type conversion functions
to directly convert one data type
to another which is useful
in day to day and competitive programming."""

x = 1

print(int(x))
print(float(x))
print(bool(x))
```

```
1
1.0
True
```

In [7]:
```
#Type conversion

#initializing string
s = "10010"
#It's still string, so it'll pring the value twice
c = s * 2
#Printing string converting to int.
d = int(s)
dz = d*2

#print string converting to float
e = float(s)

print(c)
print(dz)
print(e)
```

```
1001010010
20020
10010.0
```

In [8]:
```
#Type conversion
# Python code to demonstrate Type conversion
# using  tuple(), set(), list()

s = 'geeks'
c = tuple(s)
print(c)
print(len(c))

#Set() will remove duplicate value and it'll print in
#it's own order.

f = 'geekks'
q = set(f)
print(q)
print(len(q))

s = 'geeks'
c = list(s)
print(c)
print(len(c))
```

```
('g', 'e', 'e', 'k', 's')
5
{'k', 'g', 's', 'e'}
4
['g', 'e', 'e', 'k', 's']
5
```

In [9]:
```
#Type conversion
#Code1: Demonstrating set() with list and tuple
"""The difference between list and tuple is that the list
is mutable and tuple is immutable(you can't change the value).
"""
#initializing list
list1 = [3,4,1,4,5]

#initializing tuple
tuple1 = (3,4,1,4,5)

#printing iterables before conversion

print(str(list1))
print(str(tuple1))

#Iterables after conversion are
#notice distinct and sorted elements

print(str(set(list1)))
print(str(set(tuple1)))
```

```
[3, 4, 1, 4, 5]
(3, 4, 1, 4, 5)
{1, 3, 4, 5}
{1, 3, 4, 5}
```

In [10]:
```
#Type conversion
#Code2:Demonstration of working of set on dictionary

#Python3 code to demonstrate the working
#of set() on dictionary

#initializing list
dictionary1 = {4:'geeks', 1:'for', 3:'geeks'}

# Printing dictionary before conversion
# internaly sorted
print(str(dictionary1))

# Dictionary after conversion are
# notice lost keys
print(str(set(dictionary1)))
```

```
{4: 'geeks', 1: 'for', 3: 'geeks'}
{1, 3, 4}
```

Conditional Statements

In [11]:
```
x = 10

if x == 1:
    print("a")
elif x == 2:
    print("b")
else:
    print("c")

# # Ternary operator

# x = "a" if n > 1 else "b"

# # Chaining comparison operators
# if 18 <= age < 65:
#     print("age < 18")
```

```
c
```

List

In [ ]:
```
#List
"""
go to: https://developers.google.com/edu/python/lists

Contents:

FOR and IN
Range
While Loop
List Methods
List Build Up
List Slices
"""
```

In [12]:
```
#Creating lists

letters = ["a", "b", "c"]
matrix = [[0, 1], [1, 2]]
zeros = [0] * 5
combined = zeros + letters
numbers = list(range(20))

# Accessing items
letters = ["a", "b", "c", "d"]
letters[0]  # "a"
letters[-1] # "d"

# Slicing lists
letters[0:3]   # "a", "b", "c"
letters[:3]    # "a", "b", "c"
letters[0:]    # "a", "b", "c", "d"
letters[:]     # "a", "b", "c", "d"
letters[::2]   # "a", "c"
letters[::-1]  # "d", "c", "b", "a"

# Unpacking
first, second, *other = letters

# Looping over lists
for letter in letters:
    ...

for index, letter in enumerate(letters):
    ...

# Adding items
letters.append("e")
letters.insert(0, "-")

# Removing items
letters.pop()
letters.pop(0)
letters.remove("b")
del letters[0:3]

# Finding items
if "f" in letters:
    letters.index("f")

# Sorting lists
letters.sort()
letters.sort(reverse=True)

# Custom sorting
items = [
    ("Product1", 10),
    ("Product2", 9),
    ("Product3", 11)
]
```

```
items.sort(key=lambda item: item[1])

# Map and filter
prices = list(map(lambda item: item[1], items))
expensive_items = list(filter(lambda item: item[1] >= 10, items))

# List comprehensions
prices = [item[1] for item in items]
expensive_items = [item for item in items if item[1] >= 10]

# Zip function
list1 = [1, 2, 3]
list2 = [10, 20, 30]
combined = list(zip(list1, list2))    # [(1, 10), (2, 20)]
```

In [13]:
```
#List
"""
s = ['h','e','l','l','o']   #create a list
s.append('d')   #append to end of list
len(s)         #number of items in list
s.sort()      #sorting the list
s.reverse() #reversing the list
s.extend(['w','o']) #grow list
s.insert(1,2)   #insert into list
s.remove('d') #remove first item in list with value e
s.pop() #remove last item in the list
s.pop(1) #remove indexed value from list
s.count('o') #search list and return number of instances found
s = range(0,10) #create a list over range
s = range(0,10,2) #same as above, with start index and increment
"""


emptyList = []

list1 = ['one, two, three, four, five']

list2 = ['one', 'two', 'three', 'four', 'five']

print(list1)
print(len(list1))

print('\n')

print(list2)
print(len(list2)) # will give the length of the list
```

```
['one, two, three, four, five']
1


['one', 'two', 'three', 'four', 'five']
5
```

In [14]:
```
#List append()

media = ["movies","music","pictures"]

print(len(media))

media.append("books")
media.append("blogs")

print(media)
print(len(media))
```

```
3
['movies', 'music', 'pictures', 'books', 'blogs']
5
```

In [15]:
```
#List insert()
list = ["movies", "music", "pictures"]

list.insert(0,"files")
list.insert(2,"books")
list.insert(3,"blogs")

print(list)
```

```
['files', 'movies', 'books', 'blogs', 'music', 'pictures']
```

In [16]:
```
#List extend()

media_list = ['files', 'movies', 'books']
media_list1 = ['music', 'pictures']

media_list.extend(media_list1)

print(media_list)
```

```
['files', 'movies', 'books', 'music', 'pictures']
```

In [17]:
```
#List pop

seasons = ["summer", "winter", "spring", "fall"]
seasons.pop(0)
seasons.pop(-1)
print(seasons)
```

```
['winter', 'spring']
```

In [18]:
```
#List remove()

media_list = ['files', 'movies', 'books', 'blogs', 'music', 'pictures']

media_list.remove('files')

print(media_list)
```

```
['movies', 'books', 'blogs', 'music', 'pictures']
```

In [19]:
```
#List delete
color = ["yellow", "red", "blue"]
print(color)
shape = ["square", "triangle", "rectangle"]
print(shape)

del color[0]
print(color)
```

```
['yellow', 'red', 'blue']
['square', 'triangle', 'rectangle']
['red', 'blue']
```

In [20]:
```
#List and if statement
days = ["Sun", "Monday", "Tuesday", "Wednesday"]

if "Sun" in days:
    print("Yes")
else:
    print("No")


if "Saturday" in days:
    print("Yes")
else:
    print("No")
```

```
Yes
No
```

In [21]:
```
#List and if statement
#Keyword 'not' can be combined with 'in'

months = ['jan','feb','mar','apr']

input = "dec"

if input not in months:
    print("month you entered is not in the list")
else:
    print("month you entered is in the list")

input1 = "mar"
if input1 not in months:
    print("month you entered is not in the list")
else:
    print("month you entered is on the list")
```

```
month you entered is not in the list
month you entered is on the list
```

In [22]:
```
#List reverse
numbers = ["one","two","three","four","five"]
print(numbers)
numbers.reverse()
print(numbers)
```

```
['one', 'two', 'three', 'four', 'five']
['five', 'four', 'three', 'two', 'one']
```

In [23]:
```
#List sort
numbers = [5,4,3,2,1]
letters = ['e','d','c','b','a']

print(sorted(numbers))
print(sorted(letters))
```

```
[1, 2, 3, 4, 5]
['a', 'b', 'c', 'd', 'e']
```

In [24]:
```
#List length

mylist = "one,two,three,four,five"
mylist1 = ["one,two,three,four,five"]
mylist2 = ["one","two","three","four","five"]

print(len(mylist))
print(len(mylist1))
print(len(mylist2))
```

```
23
1
5
```

In [25]:
```
#List split

number_list = "one,two,three,four,five"
number_list1 = ["one,two,three,four,five"]
number_list2 = ["one","two","three","four","five"]

new_list = number_list.split(',')
print(new_list)
print(len(new_list))

#you can not split the number_list1 and number_list2
#because it has no attribute to split ','.
```

```
['one', 'two', 'three', 'four', 'five']
5
```

In [26]:
```
a = "My name is John. I live in Bangalore"
print(a)
print(len(a))
print('\n')

b = a.split(',')
print(b)
print(len(b))
print('\n')

c = a.split('.')
print(c)
print(len(c))
print('\n')

d = a. split(' ')
print(d)
print(len(d))
```

```
My name is John. I live in Bangalore
36


['My name is John. I live in Bangalore']
1


['My name is John', ' I live in Bangalore']
2


['My', 'name', 'is', 'John.', 'I', 'live', 'in', 'Bangalore']
8
```

In [27]:
```
#List boolean
#It'll check if all the conditions are true.
#It'll give false even one condition(s) are false.

list_boo = ['first','second','third']
list_boo[0] == 'first'
list_boo[1] == 'second'
list_boo[2] == 'third'

#Below will give the result false
# list_boo = ['first','second','third']
# list_boo[0] == 'first'
# list_boo[1] == 'second'
# list_boo[1] == 'third'
```

Out[27]:
```
True
```

In [28]:
```
#List slicing
"""variable_name[start:end] items start through end-1
variable_name[start:] items start through the rest of the array
variable_name[:end] items from the beginning through end-1
variable_name[:] whole array"

Start or end may be a negative number. It counts from the end of the
array instead of at the beginning.

a[-1] # last item in the array
a[-2:] # last two items in the array
a[:-2] # everything except the last two items
"""

z = ['yellow','green','red','blue','white']
z1 = z[1:-1]
print(z1)
```

```
['green', 'red', 'blue']
```

In [29]:
```
#List loops

items_1 = [1,2,3,4,5]

for i in items_1:
    print(i)
```

```
1
2
3
4
5
```

In [30]:
```
#List for loops incrementing value
num_val = [1,10,20,30]
cal = 1
for num_vals in num_val:
    cal = cal+num_vals
    print(cal)
```

```
2
12
32
62
```

In [31]:
```
#List for loops range
for i in range(0,5):
    print(i)
```

```
0
1
2
3
4
```

In [32]:
```
#List range

for i in range(0,3):
    print(i)
```

```
0
1
2
```

In [33]:
```
#List - Zip function - converting list into a Dictionary
# Take value from two seperate lists and make values of one list as a ke
y and second as value

#output - {1:a,2:b,3:c,4:d,5:e,6:f}

key_list = [1,2,3,4,5,6]
value_list = ["a","b","c","d","e","f"]

output_dictionary = {}

for key, value in zip(key_list,value_list):
    output_dictionary[key] = value
print(output_dictionary)

#print(len(output_dictionary))

#---
# output_dictionary = ("%d:%s" % (key,value))
# print(output_dictionary)
#----
```

```
{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f'}
```

In [34]:
```
#List - convert two lists to a dictionary.
#another way to solve the above problem.

key_list = [1,2,3,4,5,6]
value_list = ["a","b","c","d","e","f"]

print(len(key_list))

# dictt = {}

# for i in range len(key_list):
#     print(i)

#     dictt[key_list[i]] = value_list[i]
# print(dictt)
```

```
6
```

Tuples

In [12]:
```
point = (1, 2, 3)
point(0:2)     # (1, 2)
x, y, z = point
if 10 in point:
    ...
#Swapping variables
x = 10
y = 11
x, y = y, x
```

```
  File "<ipython-input-12-bd907088f62e>", line 2
    point(0:2)     # (1, 2)
             ^
SyntaxError: invalid syntax
```

Arrays

In [16]:
```
from array import array

numbers = array("i", [1, 2, 3])

#print(dir(numbers)) # Shift + Enter will give your lists of all the met
hods for arrays
```

Sets

In [ ]:
```
first = {1, 2, 3, 4}
second = {1, 5}

first | second  # {1, 2, 3, 4, 5}
first & second  # {1}
first - second  # {2, 3, 4}
first ^ second  # {2, 3, 4, 5}

if 1 in first:
    ...
```

Dictionary

In [ ]:
```
#Dictionary
#variable name = {} #to create an empty dictionary
#dictionary has key : value
#Nested dictionary - you can create a dictionary within a dictionary
```

In [17]:
```
phone_released_year = {"iphone1":2007,
            "iphone2":2008,
            "iphone3":2009,
            "iphone4":2010
            }

print(phone_released_year)

#print(dir(phone_released_year)) - it'll give you  a list of all the met
hods...
```

```
{'iphone1': 2007, 'iphone2': 2008, 'iphone3': 2009, 'iphone4': 2010}
```

In [ ]:
```
#Dictionary - add key and value to the dictionary

phone_released_year["iphone5"] = 2011

print(phone_released_year)
```

In [ ]:
```
#Dictionary - remove key and value to the dictionary
del phone_released_year["iphone1"]
print(phone_released_year)
```

In [ ]:
```
#Dictionary length
print(len(phone_released_year))
```

In [ ]:
```
#Test the dictionary
my_dictionary = {'a':'one',
                 'b':'two'}

print('a' in my_dictionary)
print('b' in my_dictionary)
print('c' in my_dictionary)
```

In [ ]:
```
#Test the dictionary using for loop

my_dictionary = {'a':'one',
                 'b':'two'}

for i in my_dictionary:
    if 'a' in my_dictionary:
        print("key found")
        break
    else:
        print("no key found")
```

In [ ]:
```
#Dictionary - get a value of a specified key..

my_dictionary = {'a':'one',
                 'b':'two'}

print (my_dictionary.get('a'))
```

In [ ]:
```
#Dictionary - print all keys with a for loop

iphones_released_years = {"iphone1":2007,
            "iphone2":2008,
            "iphone3":2009,
            "iphone4":2010
            }


print("-"*10)
print("iphones released so far:")
print("-"*10)
for model in iphones_released_years.items():
    print(model)
```

In [ ]:
```
iphones_released_years = {"iphone1":2007,
            "iphone2":2008,
            "iphone3":2009,
            "iphone4":2010
            }
for key in iphones_released_years:
    print(key)
```

In [ ]:
```
#Dictionary items
#create two variables to unpack value of items

iphones_released_years = {"iphone1":2007,
            "iphone2":2008,
            "iphone3":2009,
            "iphone4":2010
            }
for key, val in iphones_released_years.items():
    print(key,"=>", val)
```

In [ ]:
```
#Dictionary - sort dictionary
models_years = {"iphone4":2010,
            "iphone2":2008,
            "iphone1":2007,
            "iphone3":2009
            }
for keys, values in sorted(models_years.items()):
    print(keys,values)
```

Nested Dictionary

In [1]:
```
#Nested Dictionary - how to define a nested dictionary

nested_dict = { 'dict1': {'key_A': 'value_A'},
                'dict2': {'key_B': 'value_B'}}

print(nested_dict)
```

```
{'dict1': {'key_A': 'value_A'}, 'dict2': {'key_B': 'value_B'}}
```

Loops

In [ ]:
```
for n in range(1, 10):
    print(n)

    while n < 10:
    print(n)
    n += 1
```

In [ ]:
```
#Range

for d in range(1,5):
    if d == 4:
        break
    print(d)
```

In [ ]:
```
#While loop
#Depending on the use case but developer use it very rarely

while True:
    raw_input1 = input("Start typing....")
    if raw_input1 == "quit":
        break
    print(f"your answer was,{raw_input1}")
```

In [ ]:
```
#While loop

counter = 0
while counter <= 6:
    print(counter)
    counter = counter+1
```

In [ ]:
```
#While loop

counter = 0
while counter < 6:
    counter = counter+1
    print(counter)
```

In [ ]:
```
#Nested loops - loops inside the a loop
for x in range(1,3):
    for y in range(1,5):
        print(x,y)
```

In [5]:
```
#Loops through words
word = "computer"
for letter in word:
    print(letter)
```

```
c
o
m
p
u
t
e
r
```

Functions

In [ ]:
```
"""
Functions are named blocks of code designed to do
one specific job. Functions allow you to write code
once that can then be run whenever you need to
accomplish the same task. Functions can take in the
information they need, and return the information they
generate. Using functions effectively makes your
programs easier to write, read, test, and fix.
"""
```

In [ ]:
```
"""
#In Python, function is a group of related statements
that perform a specific task. Functions help break our
program into smaller and moduler chunks.

# There are two different types of functions in Python.
User-defined function and built-in functions(Functions that
readily come with python)

for more detailed content go to:
https://en.wikibooks.org/wiki/Python_Programming/Functions
"""
```

(a) User Defined Function

In [35]:
```
#Syntax of Function

"""
def function_name(parameters):
#       "docstring"
        statement(s)
  return(return statement is used to exit a function. it is optional.)
#you can return one or multiple values
# Declaring Arguments when calling function that takes
some values for furtuer processing, we need to send some
values as Function Arguments.
"""

"""
In computer programming, a parameter or a formal argument,
is a special kind of variable, used in a subroutine to refer
to one of the pieces of data provided as input to the subroutine.

For example, if one defines the add subroutine as def add(x, y):
return x + y, then x, y are parameters, while if this is called as add
(2, 3),
then 2, 3 are the arguments. Note that variables (and expressions thereo
f)
from the calling context can be arguments: if the subroutine is called a
s a = 2; b = 3;
add(a, b) then the variables a, b are the arguments, not the values 2,
 3.
See the Parameters and arguments section for more information.
"""
```

```
Hello,World
```

In [36]:
```
#Function returns a single value:

x1 = 4

def cal(x1):
    return x1*x1

cal(x1)
```

Out[36]:
```
16
```

In [37]:
```
#Function returns multiple values:

def two_items(list1):
    return list1[0], list1[1]
a, b = two_items(["Hello", "World","How","are","you?"])
print(f"{a},{b}")
```

```
Hello,World
```

In [45]:
```
# Declaring Arguments in functions

def find_max(a,b):
    if(a>b):
        print(str(a) + "is greater than" +str(b))
    elif(b>a):
        print(str(b) + "is greater than" +str(a))

find_max(30,20)
```

```
30is greater than20
```

In [ ]:
```
def increment(number, by=1):
    return number + by

increment(number,by=1)

# Keyword arguments
increment(2, by=1)

# Variable number of arguments
def multiply(*numbers):
    for number in numbers:
        print(number)


multiply(1, 2, 3, 4)

# Variable number of keyword arguments
def save_user(**user):
    ...


save_user(id=1, name="Mosh")
```

In [ ]:
```
number = 2
by = 4

def increment(number, by):
    return number + by

increment(number,by)
```

In [ ]:
```
#number = 5
def increment(number, by=1):
    return number + by

increment(number,by=1)

# Keyword arguments
increment(2, by=1)
```

(b) Built-in Functions and Methods

In [ ]:
```
#Most important built-in Python functions
#for data project
"""
#print() - it prints stuff to the screen.

input: print("Hello, World!")
output: Hello, World

#abs() - returns the absolute value of a numeric value
(e.g. integer or float). Obviously it can't be a string.
It has to be a numeric value.

input: print(-4/3)
output: -1.333333333333

round() - returns the rounded value of numeric value
input: print(round(-4/3))
output: -1

#min() - returns the smallest item of a list. It can even
be a string.

input: print(min(3,2,5))
output: 2

input: print(min('c','d','e'))
output: c

#max()

input: print(max(3,2,5))
output: 5

input: print(max('c','d','e'))
output: e

#sorted() - It sorts a list into ascending order. The list
can contain strings or numbers.

input: a = [3,2,1]
print(sorted(a))
output: [1, 2, 3]

#sum() - it sums a list.

Example1:
input: axu = [3,2,1]
sum(axu)
output: 6

Example2:
input: axu = [4/3, 2/3, 1/3, 1/3, 1/3]
sum(axu)
output:3.0000000000000004

#len() - returns the number of elements in a list
```

```
or the number of characters in a string.

input: print(len('Hello!'))
output: 6

#type()

Example1:
input: a = True
       type(a)
output: bool

Example2:
input: b = 2
       type(b)
output: int

"""
```

In [46]:
```
#List of the most important functions and methods that
#you use all the time.

#Variable Name
name_a = 'Hello World!'

#Here's a simple example of a Python function:
len(name_a)
#Result:12

#And an example for Python method:
print(name_a.upper())
#Result:HELLO WORLD!
```

```
HELLO WORLD!
```

Methods for Python Strings

In [ ]:
```
#The most important built-in Python methods for Python strings.

"""
#Methods for Python Strings

#print(dir(variable name)) - When we pass a variable in dir
it'll show us all of the attributes and metods we have access
to with that variable. But that doesn't show what any of these
actually do so to see more information about these
metods, we can use the help function. For e.g. print(help(str))
or print(help(list)), etc.

#lower() - returns the lowercase version of a string.

input: a = 'MuG'
       print(a.lower())
output: mug

#upper() - returns the uppdercse version of a string.

input: a = "hello"
       print(a.upper())
output: HELLO

#strip() - if the string has whitespaces at the beginning
or at the end, it removes them.

input:
a = '   hello  '
print(a)
print(a.strip())
output:
  hello
hello

#split() - splits your string into a list. Your argument
specifies the delimiter

input: a = 'Hello World'
print(a.split(' ')) #Note:in this case the space is the delimiter.
output:['Hello', 'World']

#join() - it joins the elements of a list into one string.

input:a = ['Hello', 'World']
print(' '.join(a))
output: Hello World

#replace() - it replaces what you want to replace with a new
             value. This method takes two arguments. First it takes
             what we want to replace, and second argument which is
             seperated by a comma is what we want to replace world with.

input: a = 'Hello World'
new_message = message.replace('world','universe') - #what we want to rep
lace
```

```
                is seperated by a comma unless you put the replacement in
 a
                new variable you won't see new value/string(because it's o
nly
                returning a new string with those vlaues replace).
                We need a new variable for new return string.
print(new_message)

#format string - {} is a place holder. For detail string formatting vide
o
go to: https://www.youtube.com/watch?v=vTX3IwquFkc

#for slicing - go to: https://www.youtube.com/watch?v=ajrtAuDg3yw
"""
```

In [27]:
```
#print(help(list))

print(help(datatype))
```

```
---------------------------------------------------------------------------
----
NameError                                 Traceback (most recent call l
ast)
<ipython-input-27-7a4e9e657f94> in <module>
      1 #print(help(list))
      2
----> 3 print(help(datatype))

NameError: name 'datatype' is not defined
```

In [ ]:
```
# To see list of all the different methods we could use on our string
gretting = 'hello'
name = 'john'
#print(dir(name)) - When we pass a variable in dir it'll show us all of
 the
#attributes and metods we have access to with that variable
```

Methods for Python Lists

In [ ]:
```
#method(arg) - arg = arguments
"""
#.append(arg) - method adds an element to the end our our list.

input: dog = ['Samosa',9,2001]
dog.append(4)
print(dog)
output:['Samosa', 9, 2001, 4]

#.remove(arg) - specify the element that we want to remove

input:dog = ['Samosa',9,2001,4]
dog.remove(2001)
print(dog)
output:['Samosa', 9, 4]

#.count(arg) - returns the number of the specified value in the list.

Example:dog.count(9)

input: dog = ['Samosa',9,9,2009]
dog.count(9)
output: 2

#.clear(arg) - removes all elements of the list. It will basically
delete the list.

input: dog = ['Samosa',9,9,2009]
dog.clear()
print(dog)
output: []

"""
```

Methods for Python Dictionaries

In [ ]:
```
"""
#.keys() - will return all the keys from your dictionary.

input: dog_dict = {'name':'Samosa',
           'age':9,
           'mob':9,
           'year':2009}
print(dog_dict.keys())
output: dict_keys(['name', 'age', 'mob', 'year'])

#.values() - will return all the values from your dictionary.

input: dog_dict = {'name':'Samosa',
           'age':9,
           'mob':9,
           'year':2009}
print(dog_dict.values())
output: dict_values(['Samosa', 9, 9, 2009])

#.clear() - will delete everything from your dictionary.

input: dog_dict = {'name':'Samosa',
           'age':9,
           'mob':9,
           'year':2009}
dog_dict.clear()
print(dog_dict)
output: {}

#*** Note:

Note:
Adding an element to a dictionary doesn't require you to
use a method; you have to do it by simply defining a
key-value pair like this:

dog_dict['key'] = 'value'
Eg.
input:
dog_dict = {}
dog_dict['name'] = 'Samosa'
dog_dict
print(dog_dict)
output:
{'name': 'Samosa'}
"""
```

Nested Functions

In [ ]:
```
"""
Nested functions are functions defined within other
functions.
"""
```

Nested Methods

In [ ]:
```
"""
https://codippa.com/nested-methods-in-python/
"""
```

API

In [ ]:
```
#Consuming data from API
"""
*** API

# API stands for Application Prgoramming Interface

# API allows one piece of software to communicate
with another piece of software. Just like human interact with
web/mobile application, instead applications are doing via API.

#There are many different kids of APIs but when people talk about
google's api or twitter's api what they are talking about it is
REST (Representational State Transfer) API.

#Usually a REST API works pretty much the same way a website does.
You make a call from client to a server and you get data back over the
http protocol

#Facebook's graph API - www.facebook.com/youtube now type
graph.facebook.com/youtube -(we made an api request in browser)
what we get back is a response to our API request in JSON format.
JSON data is a structured data organized according to key value pairs.
It'similar to excel spreadsheet you might ask for
the data that's in cell a16, you can ask json array for the data if you
want to know how many likes this Facebook page had for the data containe
d
under the key likes.

*** Parameters
# https://graph.facebook.com/youtube?fields=id,name,likes
only id, name, and likes will return because these parameters have filte
red
the data that we get out of this response.

*** Google Maps API - it allows you to take a city name or an address an
d
turn it into a set of GPS coordinates
#
"""

#Writing data to APIs

"""
#Concept of HTTP request methods. Big two are GET and POST methods.
For lists of all the methods, go to:
https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_method
s

# GET - what you use to consume data. What we saw above by passing
URL parameters in order to get data back from the API

# POST - request for writing data to the API. The best practice is to
actually put the data in the body of the request. The normal web browser
doesn't
allow you to put data in the body of a request. But you can use
Postman REST client (www.postman.com).
```

```
#Authentication - lot of websites are using for their authentication is
OAuth or OAuth2. What OAuth is doing is getting credentials, kind of lik
e
username and password although they're called a clientID and client secr
et
and then you're exchanging those for what's known as an access token and
then
you pass that access token to Twitter for e.g and Twitter knows that the
request to make the tweet is coming from you, so it sends out the tweet
 from
your Twitter account.
"""
#-------------

#Ref.: https://www.youtube.com/watch?v=7YcW25PHnAA
#For list of go to: https://www.programmableweb.com/apis/directory
```

Modules

In [ ]:

```
"""

Module is a file which contains python functions, global variables etc.
It is nothing but .py file which has python executable code/statement

#Step1: create a file 'user.py'

def welcome_message(user_name):
    message = f"Welcome {user_name}"
    return message

Step2: Import 'user' module into foo.py

import user
print(user.welcome_message("Vik"))
#print(module.method(parameter))

Step3:
From terminal run: python3 foo.py

output will be:
Welcome Vik


"""
#https://www.quora.com/What-is-the-difference-between-Python-modules-pac
kages-libraries-and-frameworks
```

Import Modules

In [ ]:
```
#Import modules -

"""
#import - Python modules can get access to code from another
module by importing the file/function using import.
for e.g.:

input: from math import sqrt  #math is a module and sqrt is a function.
In another words from module import function(or parameters)
print(sqrt(4)))
output:2.0

In the above example. The "math" module is a standard module
in Python and "sqrt" is a function which is always available.

To use mathematical functions(sqrt in above example) under this module,
you have to import the module using "import math".

For list of Functions in Python Math Module go to:
https://www.programiz.com/python-programming/modules/math

# "from sys import argv" (from the "module sys", import the
"function name or parameter argv")

#from sys(is a module that contains "system functionality")
import argv(is a list of the arguments to your program).

# import time - will import the time module

"""
```

Package

In [ ]:
```
"""
Package is namespace (In simple terms,think of a
namespace as a person's surname. It allows you to
use the same function or class name in different
parts of the same program without causing a name
collision.) which contains multiple package/modules.

It is a directory which contains a special file __init__.py

#Let's create a directory 'user'. Now this package contains
multiple packages/modules to handle user related requests.

-----------------------------
user/      #top level package
    __init__.py

    get/   #first subpackage
        __init__.py
        info.py
        points.py
        transactions.py

    create/ #secon subpackage
        __init__.py
        api.py
        platform.py
-----------------------------
Now you can import it in following way

-----------------------------
from user.get import info # imports info module from get pacakage
from user.create import api # imports api module from create package
-----------------------------

When we import any package, python interpreter searches for sub director
ies/
packages.

"""

#https://www.quora.com/What-is-the-difference-between-Python-modules-pac
kages-libraries-and-frameworks
# to read more about what is 'namespace' go to:
#https://stackoverflow.com/questions/3384204/what-are-namespaces
```

Library

In [ ]:
```
"""
It is collection of various packages. There is no difference
between package and python library conceptually.

Have a look at requests(https://github.com/psf/requests) library.
We use it as a package.

"""
#https://www.quora.com/What-is-the-difference-between-Python-modules-packages-libraries-and-frameworks
```

Framework

In [ ]:
```
"""
It is a collection of various libraries which architects
the code flow.

Let's take example of Django(https://www.djangoproject.com/)
which has various in-build libraries like Auth, user,
database connector etc.

"""
```

Web Scrapping

In [2]:
```
#!pip3 install requests
#import requests - if you don't get any errors it's working.
```

In [ ]:
```
#HTTP Response Codes (200 and 204)
"""
response code of 200, meaning,
"I did what you asked me to and everything went fine".
Response codes are always a three digit numerical code.
The response code 200 literally means "OK" and is the code
most often used when responding to a GET request.

A POST request, however, may result in code 204 ("No Content")
being sent back, meaning "Everything went OK but I don't really
have anything to show you."
"""
```

In [ ]:
```
"""
#pip3 install pipenv
#pipenv -h # -h means help. If you get a help it means it's working.
Everytime new project, you create a newfolder and created a new envriron
ment
#pipenv install requests jupyter - will download 'requests' and 'jupyte
r' libraries
and dependencies.
#pipenv run jupyter notebook - will run jupyter notebook

#pipenv install requests -html
#pipenv run jupyter notebook
#http://python-requests.org/
"""
```

In [ ]:
```
"""
WEB SCRAPPING

#To install requests-html and jupyter in virtualenv
pipenv install requests -html

#To start jupyter notebook from virtualenv
pipenv run jupyter notebook

# http://python-requests.org/

#print(response.content) - will get you the content of the website

#alway add "." before the class for e.g. ".subtext"

"""
```

Generating Random Id

In [1]:
```
import uuid
```

Python map() function

In [ ]:
```

```

Help

In [ ]:
```
"""
input: sub_downloader = SubtitleDownloader()
input: help(sub_downloader)


output:
Help on SubtitleDownloader in module __main__ object:

class SubtitleDownloader(builtins.object)
 |  An example use of this class.
 |  sub_downloader = SubtitleDownloader(verbose=True)
 |
 |
 |  sub_downloader.search_for_subtitle("fight club")
 |  sub_downloader.response_to_dict()
 |  sub_downloader.select_movie_number(movie_number=1)
 |  sub_downloader.get_subtitles_from_selected_movie()
 |  sub_downloader.download_subtitle_zip()
 |
 |  Methods defined here:
 |
 |  __init__(self, verbose=False)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  download_page_to_zip_url(self)
 |      Get the download link for the zip file from the download page.
 |
 |  download_subtitle_zip(self)
 |      get the zip url and download it.
 |
 |  download_zip_file(self, zip_url)
 |      Downloads the zip file in the current directory.
 |
 |  filter_subtitles(self, subtitle_info_list, filter_language='Englis
h')
 |      Takes all the subtitle dicts and filter them for language and ta
kes the first one.
 |
 |  get_subtitles_from_selected_movie(self)
 |      Find all the sub for selected movie and select the best one.
 |
 |  response_to_dict(self)
 |      Takes the reponse page, finds all the search results
 |      convert search results into dict and return the list of dicts.
 |
 |  search_for_subtitle(self, query)
 |      Take a query string, make the url to search and get the response
 |
 |  search_result_to_movie_info(self, single_search_result)
 |      Takes html of a search result, converts it into a dict
 |      and return the dict.
 |
 |  select_movie_number(self, movie_number=None)
 |      Let the user select a movie to download subtitle for.
 |
 |  subtitle_html_to_dict(self, single_subtitle)
```

```
 |      Takes html for single subtitle result and convert it to dict the
n return it.
 |
 |  url_to_response(self, url)
 |      takes a url, make a request and return the response
 |
 |  ----------------------------------------------------------------------
--
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)


 """
```

Object Oriented Programming

In [ ]:
```
"""

#if we take the analogy of cars
#The class is the blueprint of the car and object is the physical car
#As you can't drive the buleprint of a car, you need to create the physi
cal car according to the blueprint
but the blueprint does have all the details of the car
#class can store data and code (and logic)
#first create an object out of a class and then modify

#Python convention - whenever you define a class you define it capital l
etter
#Object will have data and logic

#If we are going to change the variable in the future we don't create it
in __init__ function
#You can create how function will behave in _init_ function.
#In Python as soon as your add 'self', the function becomes method and v
araiable becomes property
#dir(name of the object). dir(sub_downloader) or you can use help


"""
#https://www.jeffknupp.com/blog/2017/03/27/improve-your-python-python-cl
asses-and-object-oriented-programming/
```

Self

In [37]:
```
"""
self represents the instance of the class. By using the "self"
keyword we can access the attributes and methods of the class in Python.
It binds the attributes with the given arguments.

"""
```

Workflow

In [ ]:
```

```

Classes()

In [ ]:
```
"""

Classes are the foundation of object-oriented
programming. Classes represent real-world things
you want to model in your programs: for example
dogs, cars, and robots. You use a class to make
objects, which are specific instances of dogs, cars,
and robots. A class defines the general behavior that
a whole category of objects can have, and the
information that can be associated with those objects.
Classes can inherit from each other – you can
write a class that extends the functionality of an
existing class. This allows you to code efficiently for a
wide variety of situations.

"""
```

In [4]:
```
"""

class Car():

    #simple model of a car
    def __init__(self, make, model, year, fuel_capacity=15):

        #Initialize car attributes
        self.make = make
        self.model = model
        self.year = year

        #fuel capacity
        self.fuel_capacity = fuel_capacity
        self.fuel_level = 0

    def fill_tank(self, fuel_amount):
        pass

"""

"""
#For more info. go to:

https://www.jeffknupp.com/blog/2017/03/27/improve-your-python-python-classes-and-object-oriented-programming/

"""
```

In [75]:
```
class Car():

#simple model of a car
    def __init__(self, make, model, year):

#Initialize car attributes
        self.make = make
        self.model = model
        self.year = year

#fuel capacity
        self.fuel_capacity = 20
        self.fuel_level = 5

    def fill_tank(self):
#Fill gas tank to capacity
        self.fuel_level = self.fuel_capacity
        print("Fuel tank is full.")

    def drive(self):
#Simulate driving
        print("The car is moving")

#Without def __str__ output will be "<__main__.Car object at 0x1131ff9e8
>"
# but after def __str__ out will be "audi a4 2016"

    def __str__(self):
        return f'{self.make} {self.model} {self.year}'

#Modify attributes
    """You can modify an attribute's value directly, or you
    can write methods that manage updating values more carefully."""

#Modifying an attribute directly
    """
    my_new_car = Car('audi', 'a4', 2016)
    my_new_car.fuel_level = 5
    """

#Writing a method to update an attribute's value

    def update_fuel_level(self, new_level):
        """Update the fuel level"""
        if new_level <= self.fuel_capacity:
            self.fuel_level = new_level
            print(f"new fuel level is: {self.fuel_level}")
        else:
            print("The tank can't hold that much")


        return 0

#Writing a method to increment an attribute's value
    def add_fuel(self,amount):
        """Add fuel to the tank."""
```

```
        if(self.fuel_level + amount <= self.fuel_capacity):
            self.fuel_level += amount
            print(self.fuel_level)
            print('\n')
            #print("Added fuel.")
        else:
            print("The tank won't hold that much.")


#Creating an object from a class. In below example, my_car is an object

my_car = Car('audi', 'a4', 2016)

#Accessing attribute values
print(my_car.make)
print(my_car.model)
print(my_car.year)
print(my_car.make,my_car.model,my_car.year)

#Calling methods
my_car.fill_tank()
my_car.drive()

#Creating multiple objects
my_car = Car('audi', 'a4', 2016)
my_old_car = Car('maruti', 'swift', 2015)
my_current_car = Car('Toyota', 'Corolla', 2009)
print('\n')

print(my_car)
print(my_car.make, my_car.model, my_car.year)
print(my_old_car.make, my_old_car.model, my_old_car.year)
print(my_current_car.make, my_current_car.model, my_current_car.year)
print('\n')

print(my_car.update_fuel_level(5))
print(my_car.add_fuel(10))
```

```
audi
a4
2016
audi a4 2016
Fuel tank is full.
The car is moving


audi a4 2016
audi a4 2016
maruti swift 2015
Toyota Corolla 2009


new fuel level is: 5
0
15


None
```

In [64]:
```
#Increment the value (+=)

a = 20
b = 5
a += b
print(a)
```

```
25
```

Back-End/Flask

In [ ]:
```
#
"""
#pipenv install flask
#After the installation to execute the file
pipenv run python 1_simple_app.py
#Flask is a framework which is used for web development
#Framework is a code somebody has writtend which you can use to build yo
ur app.
#Flask, DJango, are frameworks. Tenser flow is ML framework.
#Flask provids you the ability to add url for your Python functions.
#First 2 lines of codes are boilder code.
#@ is a decorator. It'll attach to your function. It takes the function
 hello_world
"""
```

Loads/Dumps

In [ ]:
```
"""
# Loads gives Python dictionary
# Dumps gives gives JSON string

"""
```

Decorator

In [ ]:
```
#@ is a decorator.
#@app.route('/hello') -  for e.g.web browser www.abc.com/hello
@app.route('/hello')
def hello_world():
    return 'Hello, World!. This is Vik'
```

Boilerplate Code

In [ ]:
```
#Write the following code before you write any code.

"""
from flask import Flask
import json
app = Flask(__name__)
"""



"""
Basically you have two ways of using a
Python module: Run it directly as a script,
or import it. If you write following code and if
somebody is trying to import the code it won't work.

if __name__ == "__main__":
    app.run(debug=True)
"""
```

Json (JavaScript Object Notation) format

In [ ]:
```
"""
Whether we are accessing data of others or making our
data available to others through API, it's always
in JSON format. Computers can read JSON format.
It is used primarily to transmit data between a
server and web application."
"""

#Some important command to remember:
"""
import json
json.dumps(variable name)
"""
#
```

JSON Viewer

In [ ]:
```
https://codebeautify.org/jsonviewer
```

In [ ]:
```
#Early exit.
#Google what is early exit is in Python
```

Logging (LogGuru - Logging library)

In [ ]:
```
#Logging -  Log of whatever is happening
```

TinyDB

In [ ]:
```
#to install:

#pipenv install tinydb
```

URL Encoder

In [2]:
```
"""
Type or paste your input string to see URL encoded go to:
https://www.urlencoder.io/
"""
#https://www.urlencoder.io/
```

URL Parameters (Variables)

In [2]:
```
#URL parameters - How to pass it to the destination URL
https://support.clickmeter.com/hc/en-us/articles/211032666-URL-parameter
s-How-to-pass-it-to-the-destination-URL

#URL encoder - https://www.urlencoder.io/
"""
for e.g
Type(or paste) string here....
my name is John

URL encoded output....
my%20name%20is%20John

"""
```

HTTP Request Methods (GET and POST methods)

In [ ]:
```
"""
# HTTP Methods - GET, POST, PUT, HEAD, DELETE, PATCH, OPTIONS

# The GET Method:

GET is used to request data from a specified resource
GET is one of the most common HTTP methods

Note that the query string(name/value pairs) is sent in the
URL of a GET request: /test/demo_form.php?name1=value1&name2=value2

# The POST Method:

POST is used to send data to a server to create/update a resource.

The data sent to the server with POST is stored in the reuqest body of t
he
HTTP request:

POST /test/demo_form.php HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2

"""
```

Passing parameters (variables) in a URL

In [ ]:
```

```

Postman

In [ ]:
```
getpostman.com
```

In [ ]:
```
"""
# Postman is nothing more than a browser but with lot more functionality
that a normal browser usually will not show you

Postman is readymade API client. To access data of API.
It'll convert JSON format automatically.

CRUD = Create, Read, Update, Delete -  Basic operations we do in computi
ng.

http request builder = Method, Address, Body, Header, Cookies

http request types = C (post method) R(get method) U(put method) d(del m
ethod)

"""


"""
#Postman is ideal if you're writing your own API. If you are writing ser
ver-side
code and If you need to test your code and check how it's working. With
 Postman
you can immediately start interacting with your back-end and get some fe
edback
on how things are going, if the API is working the way you expect it to,
you can test the different parameters, different authentication methods,
setting headers, cookies whatever application needs to, etc. All these w
ithout
building client-side, and writing any front-end code.

#The only job of the postman is make request without you having to go to
a browser.

#You don't have to build your own API to take advantage of Postman. You
 can use
third party API in your application. For e.g. you can use Facebook, Twit
ter, Google
or whatever APIs are out there and check what information is available,
 how you can
interact with it, etc. You can do all of these without writing a single
 line of code.
It is a very powerful way of testing because it imporves the development
time and you know
in advance that the API is working in a certain way. So there'll be no s
urprise when you actually
start using it.

"""

#-------
#https://www.youtube.com/watch?v=FjgYtQK_zLE
#https://www.youtube.com/watch?v=FjgYtQK_zLE&t=226s
#https://www.guru99.com/postman-tutorial.html#1
```

Password Verification

In [ ]:
```
#Hash
#MD5 hastag - MD5 is an algorithm for generating hashing
#passwordgenerator
#geekforgeeks MD5 Hash in Python
#Password verification is one of the use case of hash.
#MD5 generator
```

Bootstrap / Semantic UI

In [ ]:
```
# Bootstrap - how you want something to show in html.

#www.semantic-ui.com
```

Flask

In [ ]:
```

```

DJango

In [ ]:
```
#www.djangogirls.org
```

Hasura

In [ ]:
```
"""
For creating a simple API automatically use Hasura.
"""
#hasura.io
```

Font Awesome

In [1]:
```
#www.fontawesome.com
```

Boilerplates/Templates

In [ ]:
```
#Whenever you start a project look for "flask boilerplate" or "flask coo
kiecutter"
https://github.com/cookiecutter-flask/cookiecutter-flask


#Cookiecutter creates boilerplates from templates.
https://cookiecutter.readthedocs.io/en/latest/
```

Mockup Tools

In [ ]:
```
#https://grid.layoutit.com/
#https://layoutit.com/
```

SQL Datbase

In [ ]:
```
"""

#ORM = layer on top of SQL. Advantage of ORM is you can write
Python function to call ORM and then ORM will talk to SQL. So you don't
 need to know SQL.

#sqlalchemy is one of the ORM.
#from sqlalchemy import create

#Extenstion for VS Code - SQL Lite
"""
```

Python commands

In [ ]:
```
#pipenv install....
#pipenv run....
```

Project

In [ ]:
```
#Blood donation system. In case you need a blood in emergency. There's n
o central system to donate blood.
"""

- Blood group
- City
- Contact Number
- Limit how much blood can you donate per month

#Input from user:

- System shows how much blood a person has donated in that particular mo
nth
- 2 kinds of people. Donar and Receiver
- Donar - Age, Blood Type, City, Phone #,
- Receiver - Blood Type, City

#We need to save only donar's information in the databse. Not reciever


#html form (home.html page)
#TinyDB database
#Always make the bare minimum first and then improve
#Always make sure that functionality is working




"""
```

In [ ]:
```

```