Python Cheat Sheet

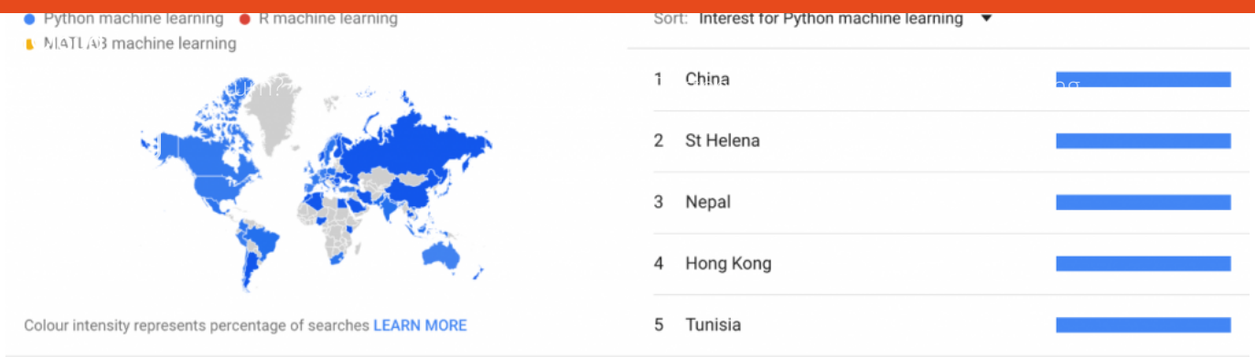
April 7, 2020

Python 3.x Cheat Sheet

Introduction

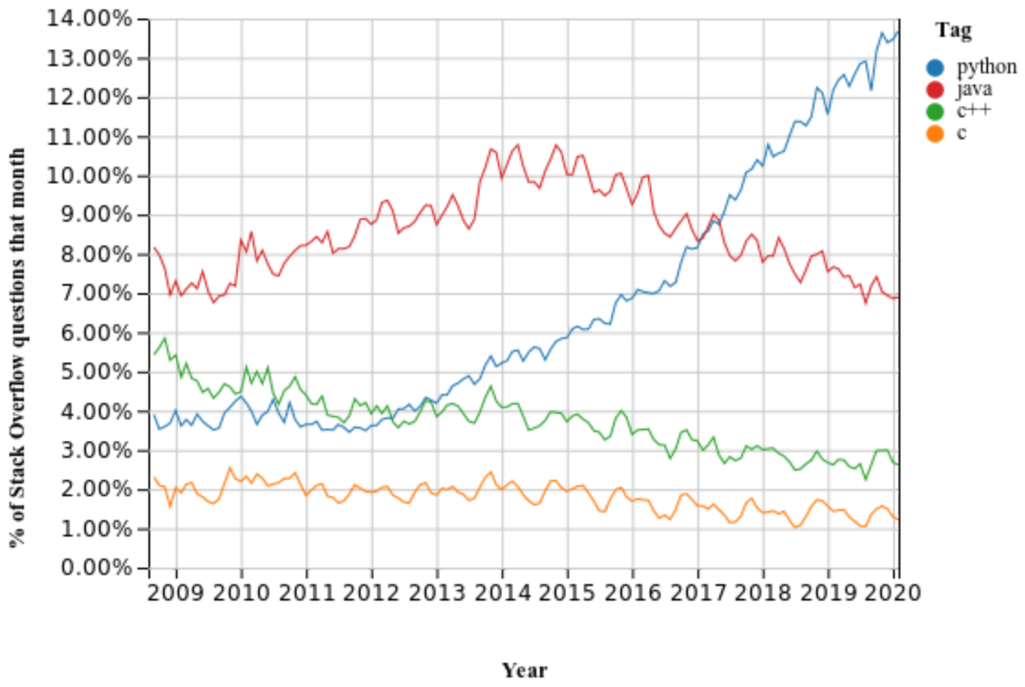
Python has rapidly become the most desired language in the job market. The wide

world.



Python's Community

The primary reason for Python's growth is its open-source development by a well-knit 30-year-old community. Trends given by StackOverflow.com are proof.



Even the corporate world loves Python. Google alone has created wildly popular machine learning libraries like Tensorflow, Keras, etc., and has distributed these for free

> We are volunteers who make and take care of the Python programming language. We have decided that January 1, 2020, was the day that we sunset Python 2. That means that we will not improve it anymore after that day, even if someone finds a security problem in it. You should upgrade to Python 3 as soon as you can.

Types

The building block in Python is an object and objects have types. A type is a way of describing the data stored inside the object. For example, to store a number we have **integer, float and complex** types. The major categories of **types** in Python are **boolean, numerics, sequences, mappings, classes** and **exceptions.** We will cover all of them in details with examples but let us start with the basic types that are used most commonly.

Basic Types

Integers

```
a = 10 # Integer
b = -9 # Integer can be negative
c = 0b011 # Integer can be Binary
d = 0o123 # Integer can be Octal
e = 0x9AF # Integer can be Hexadecimal
print("Positive Integer = ",a, " | ", type(a))
print("Negative Integer = ",b, " | ", type(b))
print("Binary Integer = ",c, " | ", type(c))
print("Octal Integer = ",d, " | ", type(d))
print("Hexadecimal Integer = ",e, " | ", type(e))
```

In Python 3 the length of an integer is limited by memory and nothing else. The type() function is used to get the type of the variable. Let's see what this code prints on the terminal. The print function converts the binary, octal and hexadecimal forms to decimal before printing

st1 = “Hello World!” #String

– The maximum value a float in Python can have is 1.8×10308. Anything bigger and Python labels it as **inf** i.e. infinity.
– Complex numbers are specified as **<**real part**>** + **<**imaginary part**>j**.
– Strings are sequences of characters. Strings are enclosed within a pair of single or double-quotes.

```
in1 =  (1+2j)  |  <class 'complex'>
fl1 =  0.78  |  <class 'float'>
fl2 =  0.026  |  <class 'float'>
fl3 =  inf  |  <class 'float'>
bl1 =  True  |  <class 'bool'>
bl2 =  False  |  <class 'bool'>
st1 =  Hello World!  |  <class 'str'>
bt1 =  b'hola'  |  <class 'bytes'>
```

String

Just like an integer, a string can be as long as one wants. To check how long the string is, we can use **len()** function. We will cover functions later, but as of now consider it a block of code that performs a single task.
str1 = “I am a string”
str2 = ‘I too am a string’
str3 = “”
str4 = str1 + ‘ ‘ + str2

```
I am a string                 | type(str1) =  <class 'str'>  | len(str1) =  13
I too am a string             | type(str2) =  <class 'str'>  | len(str2) =  17
                              | type(str3) =  <class 'str'>  | len(str3) =  0
I am a string I too am a string  | type(str4) =  <class 'str'>  | len(str4) =  31
```

– We can have a 0-length string as well. **(str3)**

Trying printing the above string without the escape characters and you will see that it

double quotes (“”) to enclose strings.

str8 = ”’Yet another way of escaping single (‘) and double (“)”’

str9 = “””Yet another way of escaping single (‘) and double (“)”””


```
Yet another way of escaping single (') and double (")
Yet another way of escaping single (') and double (")
```

Printing a Raw string reveals the escape characters used.

#Raw String

str10 = R”Let’s escape both single quote(‘) and double quote(“)”

```
Let's escape both single quote(\') and double quote(\")
```

As mentioned earlier, strings are sequences of characters. So, we can actually treat them like an array and access individual substrings using their positions. The first character of any non-empty string starts from index 0. To access any character at position L+1 in the string, we index it by writing str[L].

#Indexing String

print(“str7[0] = “, str7[0])

print(“str7[1] = “, str7[1])

print(“str7[0:10] = “, str7[0:10])

print(“str7[10:5] = “, str7[10:5])

print(“str7[10:25] = “, str7[10:25])

```
str7[0]     =  L
str7[1]     =  e
str7[0:10]  =  Let's esca
str7[10:5]  =
str7[10:25] =  pe both single
```

Note that even a space is counted as a character in a string

fl1 = 10.8


```
fl1       =  10.8   | type(fl1)          =  <class 'float'>
int(str1) =  15     | type(int(str1))    =  <class 'int'>
float(str1) = 15.0  | type(float(str1))  =  <class 'float'>
str(in1)  =  10     | type(str(in1))     =  <class 'str'>
str(fl1)  =  10.8   | type(str(fl1))     =  <class 'str'>

int(10.8)     =  10
int(-10.8)    = -10
round(10.8)   = 11
round(-10.8)  = -11
bin(10)       = 0b1010
oct(10)       = 0o12
hex(10)       = 0xa
bool(10)      = True
bool('abc')   = True
bool(None)    = False
bool('')      = False
```

– A decimal point is added when an integer is converted to a float.
– Float to integer conversion also rounds down the number.
– One can also change the representation of a number from decimal to binary etc.
– Bool() converts any non-empty string to **True** and an empty string to false.

Operators

Operators are symbols that perform an operation on one or more variables. The most common numerical operation, for example, is the addition of two or more numbers.
Numerical Operations
#Operators
num1 = 10
num2 = 4
num3 = 5e2
num1 += 1
num1 *= 3
num1 /= 2

num5 = 9

bool2 = False


```
num4            =  12
num5            =  9
bin(num4)       =  0b1100
bin(num5)       =  0b1001
bool1 or bool2  =  True
bool1 and bool2 =  False
bool1 | bool2   =  True
bool1 & bool2   =  False
bin(num4)       =  0b1100
bin(num4)       =  0b1001
num4 | num5     =  0b1101
num4 & num5     =  0b1000
num4 << 2       =  0b110000  # Number left shifted by 2 bits
num4 >> 2       =  0b11  # Number right shifted by 2 bits
```

Data Structures

Some of Python's built-in types also serve as data structures. Let us begin by looking at **list and tuples** which, just like string and byte, are **sequence types.** That is they are arranged in a sequence and can be accessed in that sequence.

List

A list can be created in one of the following ways.

– Using a pair of square brackets to denote the empty list: [] – Using square brackets, separating items with commas: [a], [a, b, c] – Using a list comprehension: [x for x in iterable] – Using the type constructor: list() or list(iterable)

List Indexing and slicing

#List Methods

l2.append(12) # Add an element at the end
sum(l2) # Sum all the elements
max(l2) # Find the maximum element
min(l2) #Find the minimum element
l2.extend(l3) # Add elements of another list to this list
l2.insert(1,13) # Insert element at a specific index
l2.remove('a') # Remove a specific element
l2.count('ab') # Count occurrences of element
l2.reverse() # Reverse List
l2.pop(4) # Remove element from this position

```
l2                                      =  [4, 1, 3, 9, 7]
l3                                      =  ['a', 'ab', 'abc', 'ab', 'ab']
Sorted l2                               =  [1, 3, 4, 7, 9]
after appending l2                      =  [1, 3, 4, 7, 9, 12]
Sum all elements of l2                  =  36
Max element of  l2                      =  12
Min element of l2                       =  1
after extending l2 with l3              =  [1, 3, 4, 7, 9, 12, 'a', 'ab', 'abc', 'ab', 'ab']
after inserting element at 1 index l2   =  [1, 13, 3, 4, 7, 9, 12, 'a', 'ab', 'abc', 'ab', 'ab']
after removing element 'a'              =  [1, 13, 3, 4, 7, 9, 12, 'ab', 'abc', 'ab', 'ab']
Count occurences of element 'ab'        =  3
Reversing the list                      =  ['ab', 'ab', 'abc', 'ab', 12, 9, 7, 4, 3, 13, 1]
Removing the element at index 4         =  ['ab', 'ab', 'abc', 'ab', 9, 7, 4, 3, 13, 1]
```

Tuple

Tuple is very similar to a list. So much so, that some people resort to calling it a cousin of the list data structure. Tuple is immutable, which means that its elements cannot be changed.

#Tuples
tp1 = ("hello",) # Single length tuple
tp2 = (1,2,3,[5,6]) # Tuple containing mutable types

```
tp2    =  (1, 2, 3, [5, 6])
tp2[0] =  1
tp2[3] =  [5, 6]
```

But what good is a variable if it can't be changed? In huge projects, which span over a

sense to have a few data objects that stay constant and can't be modified by anybody, e.g. configuration parameters etc.

Set

Set is unordered. That means it doesn't preserve the order in which elements are added to the set. Therefore, it doesn't support sequence methods like indexing or slicing. A set is used to remove duplicates, and computing mathematical operations such as intersection, union and difference.

#Sets

set1 = {'a', 'b', 'c', 'a'}

set2 = {'a', 'd', 'e', 'f'}

set3 = {'a', 'd'}

```
set1 =  {'b', 'c', 'a'}  | type(set1) =  <class 'set'>  | len(set1) =  3
set1.union(set2) =  {'e', 'a', 'd', 'f', 'c', 'b'}
set1.intersection(set2) =  {'a'}
set1.issubset(set2) =  False
set3.issubset(set2) =  True
```

– Note that each element is unique in the set.

– The order of elements in a set is different every time it is printed.

list1 = ['a','b', 'a', 'a'] set1 = **set**(list1) #How to convert list to a set

– Note how a set is created from a list.

– Also, this serves as a method to find out the unique elements in a list.

```
list1 =  ['a', 'b', 'a', 'a']  | list1 length =  4
set1  =  {'b', 'a'}            | set1 length  =  2
```

Dictionary

What is known as a hash in other languages is called a dictionary in Python. A dictionary is

Dictionaries are very useful because they allow fast searching. Secondly, most web

which is essentially a dictionary.

Control Flow

In Python, the flow of a program can be controlled by various statements. These statements alter the flow based on certain Boolean conditions.

If/Else

If/else is the most common control flow statement. Elif is short for 'else if'.

```
#Conditionals – If/Else
count = 26
if(count < 10):
   print("Too Less")
elif(count >= 10 and count < 30 ):
   print("Just right")
else:
   print("Too much")
```

```
Just right
```

For Loop

For loops are used to repeat a certain block of code a predetermined number of times. Unlike C or Java, the for loop in Python loops over sequences like list, string or even a structure like dictionary as well. Let's take a look at some examples.

```
#For Loop
list1 = [1,2,3,4,5,6,7,8,9,10] count = 0
print("list1 = ", list1)
for list_item in list1:
print("list1[",count,"] = ", list_item)
count += 1
```

```
Hello World!
```

In case we want to break the loop upon some special condition, we use the **break statement.** In the code block below, we will loop over a list of fruits. We know that an elephant has snuck into our fruit basket and we want to remove it. So as soon as we find elephant, we halt our loop and stop printing any more fruits.

```
fruits = [“apple”, “banana”, “cherry”, “dragonfruit”, “elephant”, “fig”] print(“Available Fruits = “, fruits)
for fruit in fruits:
  if(fruit == “elephant”):
    print(“I cannot eat”, fruit)
break
  else:
    print(“I can eat”, fruit)
```

```
Available Fruits =  ['apple', 'banana', 'cherry', 'dragonfruit', 'elephant', 'fig']
I can eat apple
I can eat banana
I can eat cherry
I can eat dragonfruit
I cannot eat elephant
```

A **Continue** statement is used to skip everything after itself and move on to the next iteration of the loop, whereas **Pass** is just a filler statement.

```
# Pass Statment
Current Letter : r
Current Letter : t
This is pass block
Current Letter : h
Current Letter : d
Current Letter : a
Current Letter : y

# Continue Statment
Current Letter : B
Current Letter : i
Current Letter : r
Current Letter : t
Current Letter : d
Current Letter : a
Current Letter : y
```

While Loop

The difference between **for** and **while** is that while keeps on going until its condition turns false.

#While Loop
count = 0
while count < 5:
print(count)
count += 1

```
0
1
2
3
```

time we change the upper_limit, we can write a function.


```
def generate_number_list(upper_limit):
  index = 0
output_list = [] #Empty List
  while(index < upper_limit):
output_list.append(index)
    index+=1
  return output_list
print(generate_number_list(5))
print(generate_number_list(10))
```

– Upper_limit is a parameter of the function
– To output the generated list we use a keyword **return**
– Using the same function with a different parameter(argument) we could generate two different lists

```
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Named Arguments Vs Positional Arguments

If we have multiple parameters, we may use positional arguments or named arguments.
– Positional arguments are mentioned in the order they are defined in the function.
– Named orders are defined like this: argument_name = argument value, and they can follow any order.

```
#Named Argument vs Positional Argument
def generate_number_list(start_index, upper_limit):
 index = start_index
output_list = [] #Empty List
 while(index < upper_limit):
output_list.append(index)
```

```
    while(index < upper_limit):
        index = index + increment
    return output_list
print("Default Value for Increment = ",generate_number_list(0,11))
print("Non-Default Value for Increment = ",generate_number_list(0,11,2))
```

```
Default Value for Increment         =  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Non-Default Value for Increment     =  [0, 2, 4, 6, 8, 10]
```

Lambda Functions

Python allows us to create functions without a name. These functions do not contain the **def** keyword that we saw above. However, they contain the keyword **lambda,** and so they are called lambda functions. They have the following structure: **lambda arguments : expression.**

```
#Lambda Function
x = lambda a : a/2
```

```
x(100)     =  50.0
x(x(100)) =  25.0
```

Range

Rather than being a function, **range** is actually an immutable sequence type like a tuple. It generates a range of numbers.

```
#Range Function
list1 = [] for x in range(5):
list1.append(x)
print("list1 = ", list1)
list2 = [] for x in range(2,5):
list2.append(x)
```

Python contains a huge list of other built-in methods. Check those out here and try some

Classes

Python is an object-oriented programming language and classes are the foundation of an object-oriented design. Objects are instances of a class. Any action that is performed on these objects is done through class methods. Let's take a look at how to create and use a class.

Creating Class and Methods

```
#Import Random Library
import random
#Class Definition
class Ball:
''' A simple ball class to demonstrate OOP'''
#Constructor Function
 def __init__(self, brand, age, color):
self.brand = brand
self.age = age
self.color = color
#Game for which ball is required and its speed in miles/hour
self.game = "Football"
self.speed = 0
#Method to kick the ball
 def kick(self):
   self.speed = random.randint(5,50)
print("Ball ",self.brand," has been kicked. Its new speed is ",      self.speed, " miles/hour")
 def stop(self):
   if(self.speed == 0):
print("The ",self.brand," ball is already stopped")
else:
    self.speed = 0
```

```
print("Ball2 brand | age |color = (",ball2.brand, " | ", ball2.age, " | ", ball2.color,")")
ball1.stop()
print( "Ball2 speed = ", ball2.speed)
ball2.kick()
print("Ball2 speed = ", ball2.speed)
ball2.stop()
print("Ball2 speed = ", ball2.speed)
```

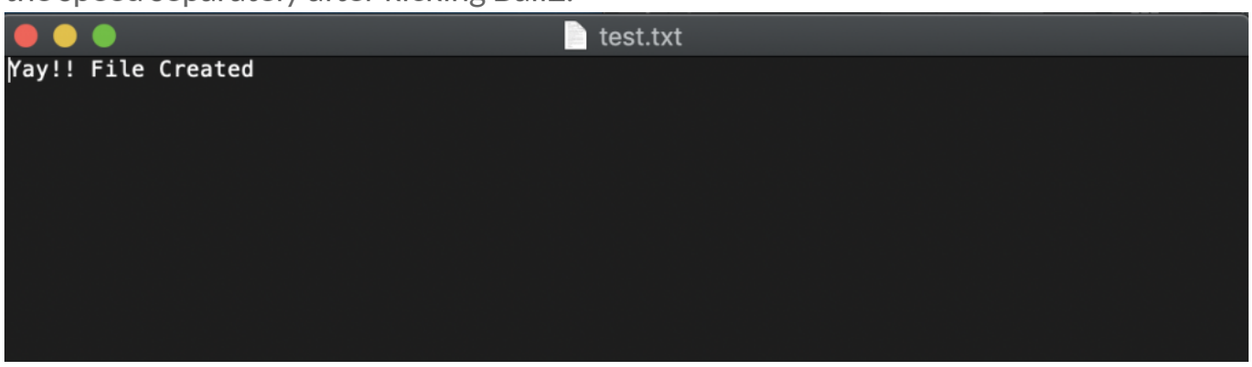
```
Ball1 brand | age |color = ( Lego  |  10  |  Black )
Ball2 brand | age |color = ( Nike  |  1  |  Yellow )

Ball1 speed =  0
The  Lego  ball is already stopped

Ball2 speed =  0
Ball  Nike  has been kicked. Its new speed is  14  miles/hour
Ball2 speed =  14
Nike  Ball successfully stopped
Ball2 speed =  0
```

Notice that the same method is used to create both the balls. Just the arguments are different.

- We can access the attribute of an object by writing object.attribute (e.g. Ball2.color gives us its color).
- We can call a method by writing object.method() e.g. Ball2.kick()
- The kick method modifies the speed attribute of the ball. We checked that by printing the speed separately after kicking Ball2.



Python is a community-driven language. A library exists for every possible problem in the

you can import a library, you need to install it using Pip. **Pip** is a package installer that comes bundled with Python.

PIP

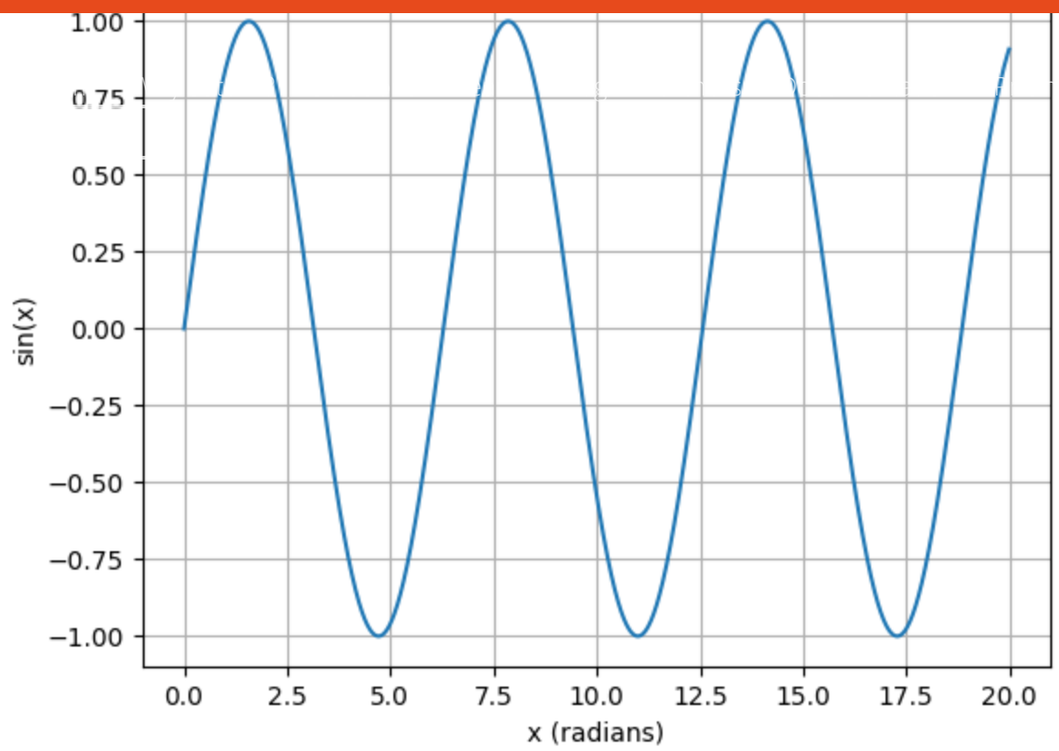
To install any package, we just type
pip install package_name.
So, let's say we wanted to install a package called **PyGame.** It is a library to make games using Python. The following would happen once we run pip for this package.

```
$ pip install pygame
Collecting pygame
  Downloading pygame-1.9.6-cp37-cp37m-macosx_10_11_intel.whl (4.9 MB)
     |████████████████████████████████| 4.9 MB 1.9 MB/s
Installing collected packages: pygame
Successfully installed pygame-1.9.6
```

Matplotlib

Matplotlib is a package to create data visualizations.

```
#matplotlib (Plotting a sine wave)
import matplotlib
import matplotlib.pyplot as plt
import math
# Data for plotting
x_axis = range(0,2000)
x_axis = [x/100 for x in x_axis] y_axis= [math.sin(x) for x in x_axis] fig, ax = plt.subplots()
ax.plot(x_axis, y_axis)
ax.set(xlabel='x (radians)', ylabel='sin(x)',
title='How Sine Wave Looks Like')
ax.grid()
fig.savefig("test.png")
plt.show()
```



Pandas

Pandas is a key Python library for data scientists. It has a data structure called DataFrame, which is like a table of data. With Pandas we can read csv/xls files and store the data in DataFrames. In this example, we will import housing data from a csv, store it in a dataframe, and analyze that data.

Head() function shows you the first 5 rows of your data. Each row is the data of a particular household in California.

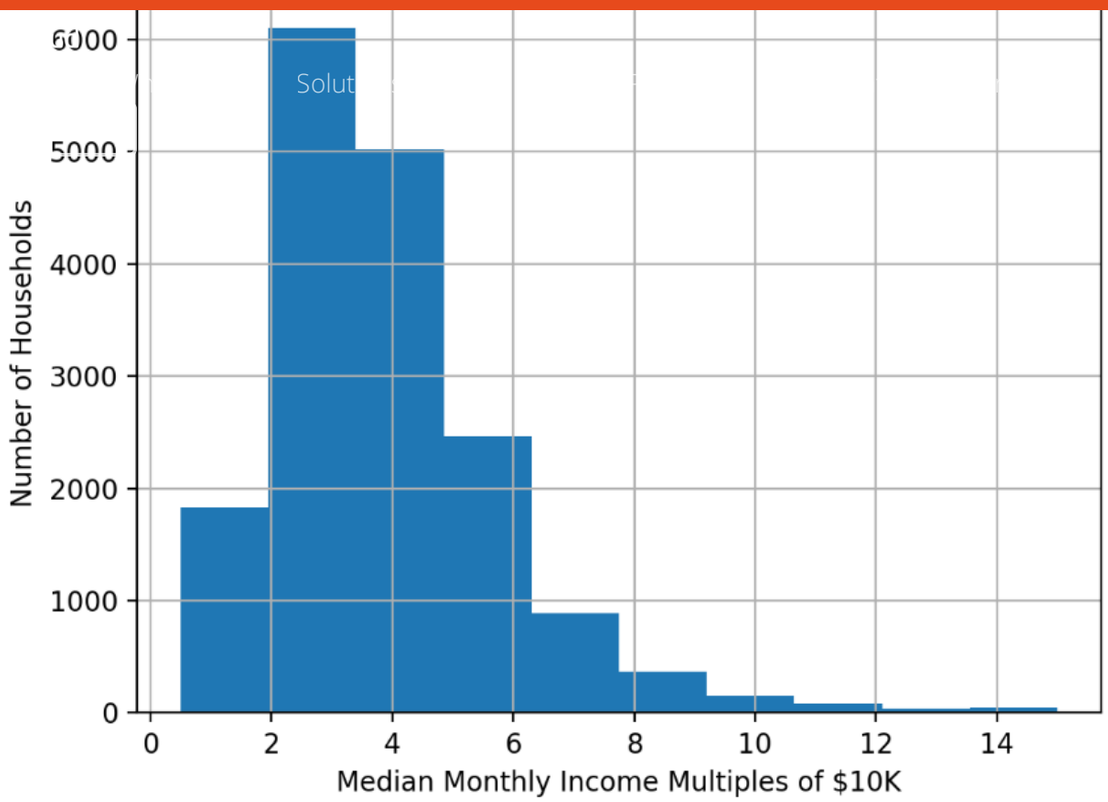
import numpy **as** np

import pandas **as** pd

house_data = pd.read_csv("https://download.mlcc.google.com/mledu-datasets/california_housing_train.csv", sep="")

print(house_data.head())

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0



Beautiful Soup – The Package to Scrape Web

Beautiful soup package helps you scrape html from any webpage and break it down into its various components. For example, let's say we want to fetch the names of all the faculty members of English Department at Cambridge University. We could write the code below.

```
#Beautiful Soup
import requests
from bs4 import BeautifulSoup
import re
URL = "https://www.english.cam.ac.uk/people/"
r = requests.get(URL)
soup = BeautifulSoup(r.content)
```

from those links.

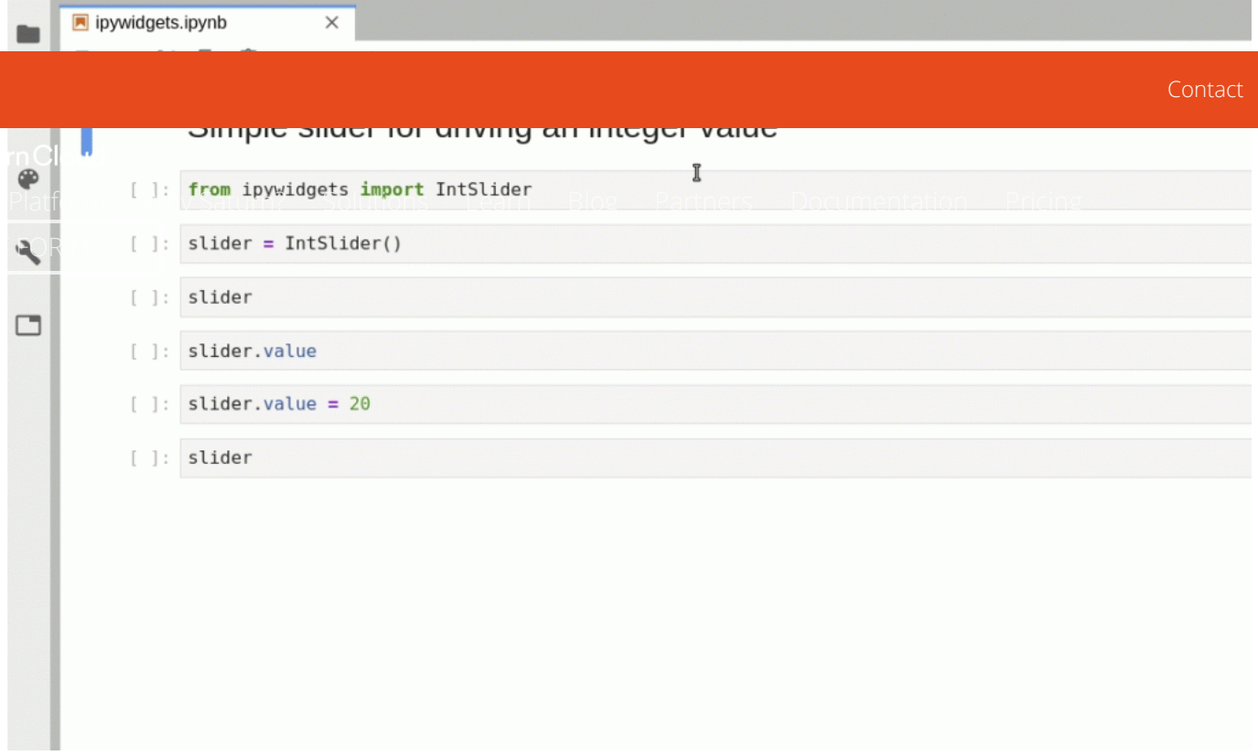

```
Dr Rebecca Anne Barr
Dr Caroline Bassett
Dr Jenny Bavidge
Dr Joanna Bellis
Dr Jessica Benenbeim
Dr Russia Boddy
Dr Deborah Bowman
Dr Christopher Burlinson
Dr Ian Burrows
Ms Sarah Cain
Dr Hero Chalmers
Dr Paul Chirico
Dr David Clifford
Dr Philip Connell
Prof Steve Connor
Dr Alexandra da Costa
Dr Orietta Da Rold
Dr Laura Davies
Prof Peter De Bolla
Dr Tania Demetriou
Dr Sarah Dillon
Dr Katrin Ettenhuber
Dr Tamara Follini
Dr Michele Gemelos
Dr Caroline Gonda
Dr Priyamvada Gopal
Dr Mina Gorji
Dr Fiona Green
Dr Sarah Haggarty
Dr Paul Hartle
Dr Anna-Maria Hartmann
Dr David Hillman
Dr Alex Houen
Dr Sarah Houghton-Walker
Dr Michael Hrebeniak
Dr Jane Hughes
Dr Michael Hurley
Dr Ewan Jones
Dr Louise Joy
Dr James Kelly
```

Python's immensely rich database of packages makes it a very powerful language. I encourage you to explore

other packages as well.

Jupyter

Finally, in the world of AI and Data Science, Jupyter is increasingly being the go-to tool for presenting your work at conferences or meetings. You can present your Python code in a



Conclusion

Python language is the Thor's hammer in today's world. It is easy to learn, quick to master and amazing to experiment with. It is a must-have skill to be adept at Python. Just pick up any good tutorial and start converting your coffee to code.

Guest post: Rishi Sidhu

Stay up to date with Saturn Cloud on LinkedIn and Twitter.
You may also be interested in: 10 Tips to Save You Time and Frustration When Programming

What is Python?

Python is a high-level, general-purpose, readable programming language that allows users

through an interpreter. Functions in Python have no restrictions on type of argument or return value. Python is a dynamically-typed language and is well known as the easier language to maintain, as it is simpler to use and it is object-oriented.

C++ is more challenging to code in, and it doesn't support garbage collection or rapid prototyping. It has more restrictions on functionality, is a statically-typed language, and is much less clean and manageable than Python.

Is Python free?

Python is free, open-source, and available for anyone to download and install at python.org. Its huge and growing ecosystem has a wide array of open-source libraries and packages that are available for all users.