TECH NOTE

Python Cheat Sheet

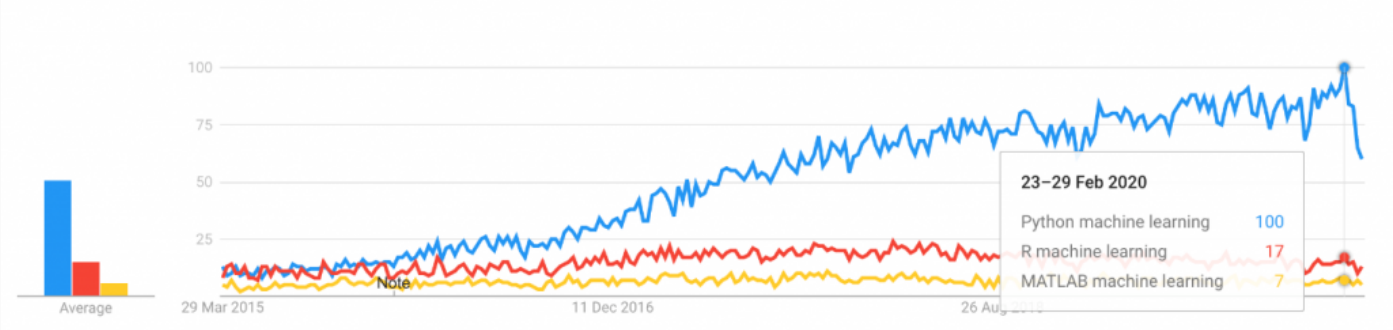
Saturn Cloud

Learn the basic applications of what has rapidly become the most desired language in the job market.

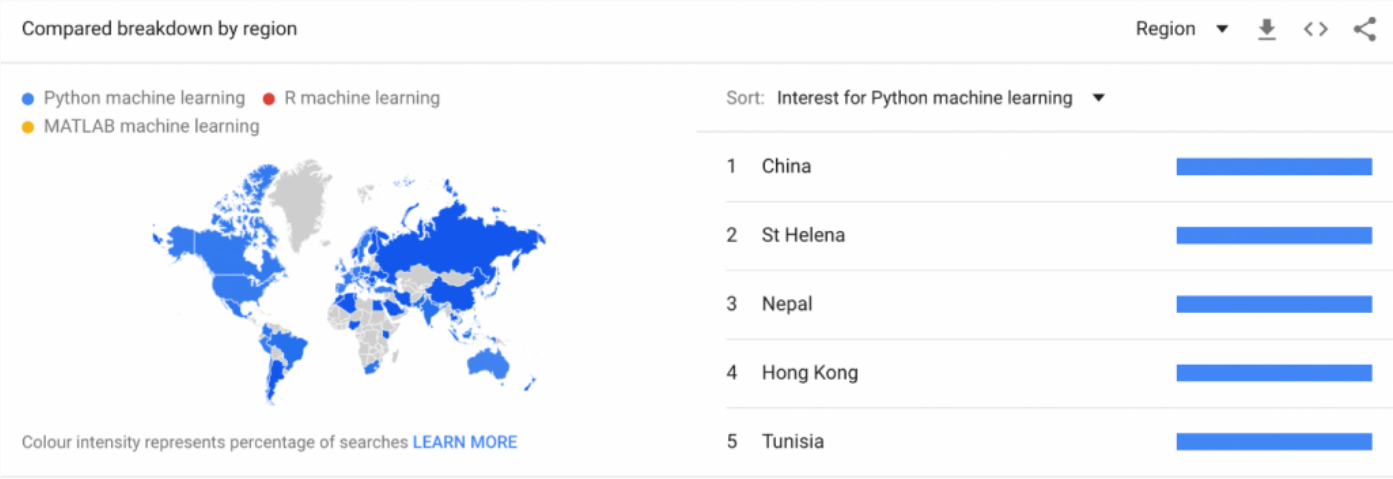
SECTIONS

INTRODUCTION

Python has rapidly become the most desired language in the job market. The wide applicability of Python, from web development to machine learning, is the reason why Python programmers are in high demand. A simple comparative analysis with the other machine learning languages shows its increasing popularity on Google Trends.



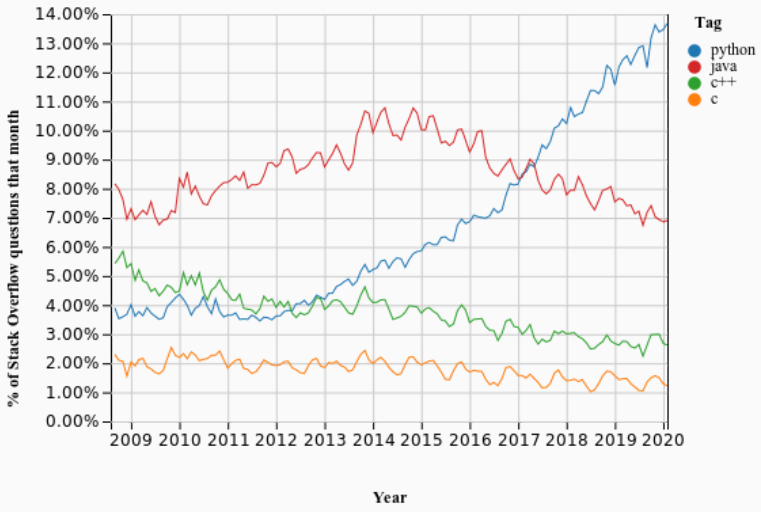
Python's worldwide dominance is also shown by how frequently it is searched across the world.



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 those for free.

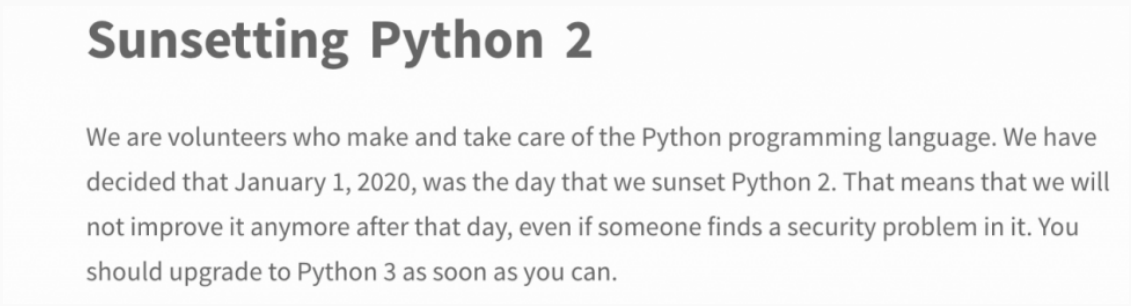


TECHNICALLY SPEAKING

Python is:

- Interpreted
- Platform Independen
- Embeddable
- Dynamically-typed

This means that Python is interpreted directly into machine-understandable bytecode unlike C, C++, or Java, which require separate compilation and execution steps. It is due to these properties that Python runs on everything from digital watches to the Mars Rover.In this cheat sheet we will be focusing on only Python 3 because on January 1st, 2020, the community decided to stop supporting Python 2 any further. This event was called Sunsetting Python 2.



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.

```
Positive Integer =  10  |  <class 'int'>
Negative Integer =  -9  |  <class 'int'>
Binary Integer =  3  |  <class 'int'>
Octal Integer =  83  |  <class 'int'>
Hexadecimal Integer =  2479  |  <class 'int'>
```

Complex, Float, Boolean, String and Byte

The other basic types are listed below.

```
in1 = 1+2j # Complex Number
fl1 = 0.78 # Float
fl2 = 2.6e-2 #Float with scientific notation. e means 10^
fl3 = 1.8*10**308 +1 #One greater than the maximum float
bl1 = True #Boolean True
bl2 = False #Boolean False
st1 = “Hello World!” #String
bt1 = b”hola” #Byte
```

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)
- The quotes are called delimiters.
- We can combine two strings that use different delimiters. (str4)

To include a delimiter in a string, enclose the string within the other kind of delimiter.
str5 = “Let’s say we need a single quote i.e. -> ‘ ”
str6 = ‘What if we “need” a double quote i.e. -> ” ‘

```
Let's say we need a single quote i.e. -> '
What if we "need" a double quote i.e. -> "
```

JIf we need both kinds of quotes inside the string, we use escape characters. Escape characters take away the functionality of the character and render that character verbatim. In Python 3, backslash “” is an escape character.str7 = “Let’s escape both single quote(‘) and double quote(“)”

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

Trying printing the above string without the escape characters and you will see that it throws an error. Another way to escape is by using triple single-quotes (‘’’) or triple 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.

String

Types can be interconverted. Interconversion is useful in applications like web development where you might have to store server statistics as strings. So, you might need to convert integers and boolean values to strings.

#Type Conversion

str1 = “15”

in1 = 10

fl1 = 10.8

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

```
num1              =  10
num2              =  4
num3              =  500.0
num1 +  num2      =  14
num1 -  num2      =  6
num1 *  num2      =  40
num1 /  num2      =  2.5     # Quotient
num1 // num2      =  2       # Floored Quotient
num1 % num2       =  2       # Remainder
num1 ** num2      =  10000   # Raised to the power in python is written as **
pow(num1,num2)    =  10000   # Raising to the power in a different manner
num1 += 1 result  =  11
num1 *= 3 result  =  33
num1 /= 2 result  =  16.5
```

The floored quotient rounds down the result of the division. Also, notice the three combined assignment statements at the bottom. These are respectively called **addition assignment, multiplication assignment**, and **division assignment** operators.

Bitwise Operations

Bitwise operations make sense for integers only. Special attention must be paid to | and & operators. To clearly see them at work, it is important to print your integers as binary numbers.

#Bitwise Operations

num4 = 12

num5 = 9

bool1 = True

bool2 = False

```
num4              =  12
num5              =  9
bin(num5)         =  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

Indexing means picking up a specific element at an index/position.
Slicing means picking up a subset of the list.
#List Indexing
l1 = [1,2,3,4,5,6,7,8,9,10]

```
l1       =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  | type(l1) =  <class 'list'>
l1[0]    =  1
l1[1]    =  2
l1[0:5]  =  [1, 2, 3, 4, 5]                  # Pick all elements from 0th position to 4th position
l1[:5]   =  [1, 2, 3, 4, 5]                  # Same as above
l1[::2]  =  [1, 3, 5, 7, 9]                  # Pick Every second element starting with 0th position
l1[::4]  =  [1, 5, 9]                        # Pick Every fourth element starting with 0th position
l1[1::4] =  [2, 6, 10]                       # Pick Every fourth element starting with 1st position
l1[-1]   =  10                               # Pick the last element
l1[:-1]  =  [1, 2, 3, 4, 5, 6, 7, 8, 9]      # Pick till 1 before the last element
l1[::-1] =  [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]  # Reverse the list
```

List[0:5] picks out the first 5 elements of the list. This operation is called slicing and is a peculiar feature of sequence types. Later we will see how this doesn't apply to unordered types like sets and dictionaries.

List Methods

#List Methods
l2 = [4,1,3,9,7] l3 = ['a','ab', 'abc','ab', 'ab'] l2.sort() # Sort
l2.append(12) # Add an element at the endsum(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 elementl2.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]
```

- Tuple is enclosed in circular brackets.
- A single element tuple needs a comma to distinguish it from a regular string.
- We can change an element of the tuple if it is a list (or any other mutable type for that matter).

#Immutability
tp2[0] = 0

```
    tp2[0] = 0
TypeError: 'tuple' object does not support item assignment
```

T#Changing the mutable partst

p2[3][1] = 4

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

After changing the list element in tuple, tp2 =  (1, 2, 3, [5, 4])
```

But what good is a variable if it can't be changed? In huge projects, which span over a hundred thousand lines of code and are modified by hundreds of programmers, it makes 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 a mapping type. It is also unordered but it is mutable. A dictionary is made up of key:value pairs. The keys should all be hashable and unique.

An object is hashable if it has a hash value which never changes during its lifetime, and can be compared to other objects. Hashable objects which compare equal must have the same hash value.

```
#Dictionary
dict1 = {‘a’: 12, ‘b’: 13, ‘c’:14, ‘d’:15}
dict2 = {‘d’ : 16, ‘e’: 17}
# Update old key and add new key
dict1.update(dict2)
```

```
dict1                 =  {'a': 12, 'b': 13, 'c': 14, 'd': 15}
len(dict1)            =  4
dict1.keys()          =  dict_keys(['a', 'b', 'c', 'd'])
dict1.values()        =  dict_values([12, 13, 14, 15])
dict2                 =  {'d': 16, 'e': 17}
dict1 after update    =  {'a': 12, 'b': 13, 'c': 14, 'd': 16, 'e': 17}
```

Dictionaries are very useful because they allow fast searching. Secondly, most web development frameworks use APIs these days. The basic data structure in APIs is JSON 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 = 0p
rint("list1 = ", list1)
for list_item in list1:
print("list1[",count,"] = ", list_item)
count +=1
```

```
list1      =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
list1[ 0 ] =  1
list1[ 1 ] =  2
list1[ 2 ] =  3
list1[ 3 ] =  4
list1[ 4 ] =  5
list1[ 5 ] =  6
list1[ 6 ] =  7
list1[ 7 ] =  8
list1[ 8 ] =  9
list1[ 9 ] =  10
```

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 = 0p
rint("list1 = ", list1)
for list_item **in** list1:
print("list1[",count,"] = ", list_item)
count +=1

```
list1      =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
list1[ 0 ] =  1
list1[ 1 ] =  2
list1[ 2 ] =  3
list1[ 3 ] =  4
list1[ 4 ] =  5
list1[ 5 ] =  6
list1[ 6 ] =  7
list1[ 7 ] =  8
list1[ 8 ] =  9
list1[ 9 ] =  10
```

```
Fstr1 = “Hello World!”
print(str1)
for c in str1:
    print(c)
```

```
Hello World!
H
e
l
l
o

W
o
r
l
d
!
```

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.

Continue is specific to loops

In the example below, see how pass does nothing, whereas continue skips the print statement as well as the letter **h**.

```
for letter in 'Birthday':
if letter == 'h'
:pass
        print('This is pass block')
    print('Current Letter :', letter)
for letter in 'Birthday':
    if letter == 'h':
continue
        print('This is continue block')
    print('Current Letter :', letter)
```

```
# Pass Statment
Current Letter : B
Current Letter : i
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
4
```

Functions

When a particular task needs to be done again and again, it is wrapped inside a **function**. Functions which are attached to objects of a class are called **instance methods**. The other types of methods are the **built-in method**s. These are the ones that are attached to specific types e.g. len() attached to lists, dicts and sets. Functions that we have seen till now – len(), count(), print(), sort(), append(), insert(), remove() etc. are all examples of built-in methods. A few properties of functions:

- They may accept inputs (called parameters).
- They may return information that is generated inside.

Let's say we want to generate a list of numbers using a while loop. The list starts at number 0 and continues till the number upper_limit. Instead of writing a while loop every time we change the upper_limit, we can write a function.

```
#Functions
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)
        index+=1
    return output_list
print("Positional Arguments = ",generate_number_list(1,11))
print("Named Arguments = ",generate_number_list(upper_limit = 10, start_index = 2))
```

```
Positional Arguments =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Named Arguments      =  [2, 3, 4, 5, 6, 7, 8, 9]
```

Default Arguments

Sometimes we may have some parameters that only appear occasionally. So, we provide a default value for them in the function definition in case they don't appear.

```
#Default Arguments
def generate_number_list(start_index, upper_limit, increment=1):
        index = start_index
output_list = [] #Empty List
        while(index < upper_limit):
output_list.append(index)
        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)
print("list2 = ", list2)
list3 = [] for x in range(2,10,3):
list3.append(x)
print("list3 = ", list3)
#Directly creating list from range
print("list(range(10)) = ", list(range(10)))
```

```
list1 =  [0, 1, 2, 3, 4]
list2 =  [2, 3, 4]
list3 =  [2, 5, 8]
list(range(10)) =  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```


```
list1 =  [0, 1, 2, 3, 4]
list2 =  [2, 3, 4]
list3 =  [2, 5, 8]
list(range(10)) =  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```


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

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/hourself.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 = 0print(self.brand," Ball successfully stopped")
```

- Keyword class creates the class. Note that convention dictates that the class name should begin with a capital letter
- __init__ is a constructor function and it is this function which will create the object when we call Ball()
- This class has five attributes, or fields, or properties named brand, age, color, game and speed.
- This class has 2 instance methods called kick and stop.
- Kick method kicks the ball and gives it a random velocity.
- Stop method makes the velocity 0 or throws a warning if the ball is already not moving.

Creating Objects

```
#Creating Objects from Ball Class
ball1 = Ball(“Lego”, 10, “Black”)
ball2 = Ball(“Nike”, 1, “Yellow”)
print(“Ball1 brand | age |color = (“, ball1.brand, ” | “, ball1.age, ” | “, ball1.color, “)”)
print(“Ball2 brand | age |color = (“,ball2.brand, ” | “, ball2.age, ” | “, ball2.color,”)”)
print(“Ball1 speed = “, ball1.speed)ball1.stop()
print(“Ball2 speed = “, ball2.speed)ball2.kick()print(“Ball2 speed = “, ball2.speed)ball2.stop()
print(“Ball2 speed = “, ball2.speed)
```

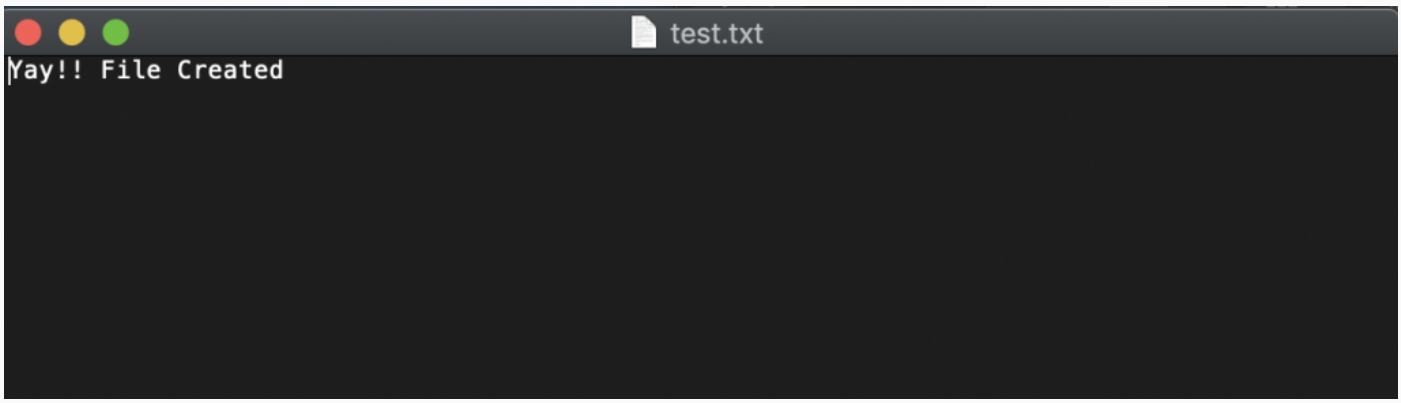
```
Ball1 brand | age |color = ( Lego  |  10  |  Black )
Ball2 brand | age |color = ( Nike  |  1  |  Yellow )

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.



File Handling

Python provides inbuilt methods to read and write files. To create and write to a text file, we call Python's inbuilt function open. We pass the name of the file as a parameter. Another parameter dictates the mode in which the file will be opened.

"a" – Append – To append to the end of the file

"w" – Write – To overwrite any existing content

"r" – Read – To read the contents

#File Handling

filename = "test.txt"

file_handle = open(filename, "w")

file_handle.write("Yay!! File Created")

#File Handlingfile_handle = open(filename, "r")

- print(file_handle.read())

LIBRARIES/MODULES

Python is a community-driven language. A library exists for every possible problem in the world. To use any external library in your program, you have to import it. However, before you can import a library, you need to install it using Pip. Pip is a package installer that comes bundled with Python.

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.

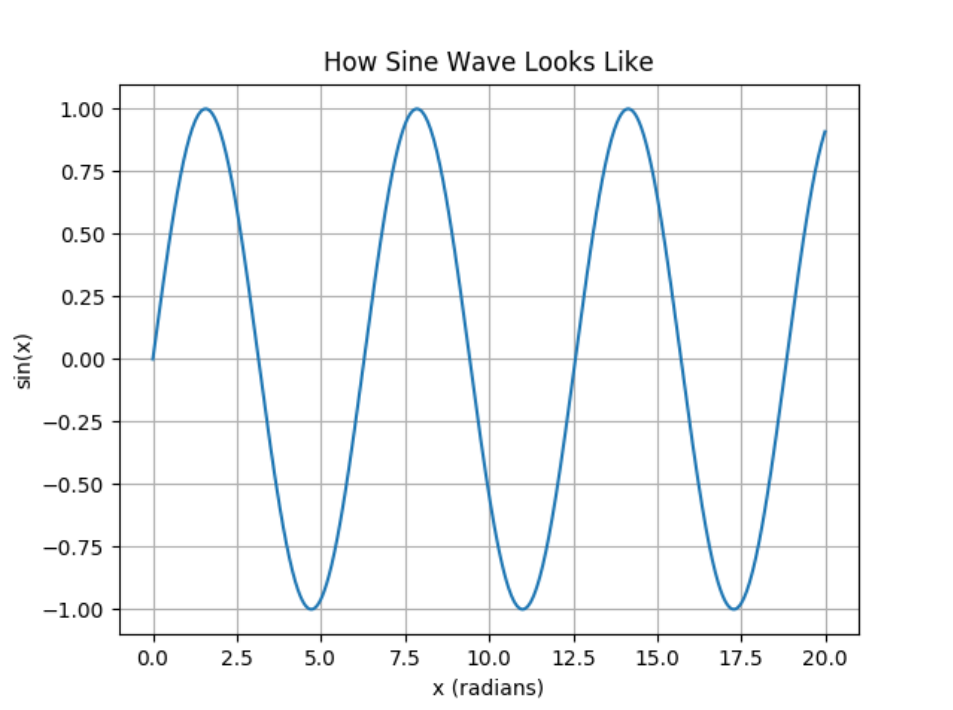
```
$ pip install pygame
Collecting pygame
  Downloading pygame-1.9.6-cp37-cp37m-macosx_10_11_intel.whl (4.9 MB)
     |████████████████████████████████| 4.9 MB 1.9 MB/s
Installing collected packages: pygame
Successfully installed pygame-1.9.6
```

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()
```

Look at how simple is it to import the module and start using its methods. For example, to compute sine values, we imported Math module and just used math.sin()



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 =

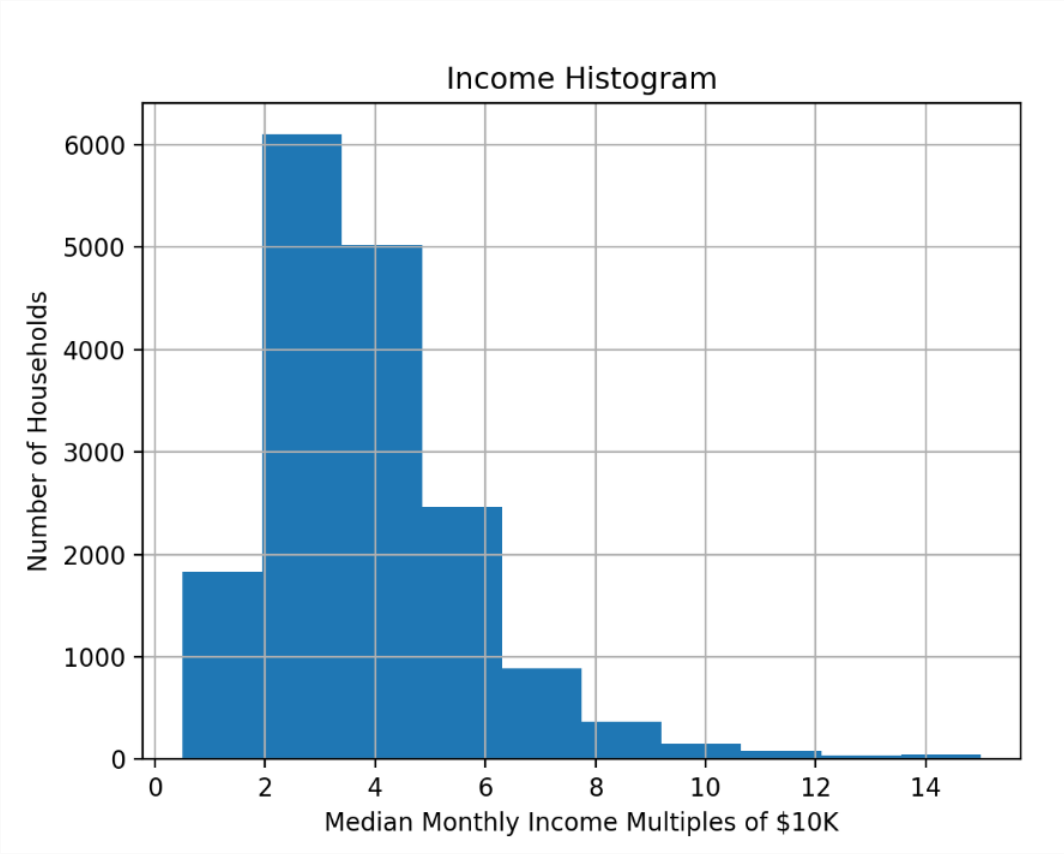
pd.read_csv("https://download.mlcc.google.com/mledu-datasets/california_housing_train.csv", sep=",")

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
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250	65500.0

The second from the right column in the dataframe above points to medium-income values for a given household. It makes sense to see how the income is distributed across all households. So we will attempt to draw a histogram of income using the **hist()** method of dataframes.

```
import matplotlib.pyplot as plt
house_data[“median_income”].hist()
plt.show()
```



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
)for link in soup.findAll('a', attrs = {'href':
re.compile("^/people/")}):
print(link.string)
```

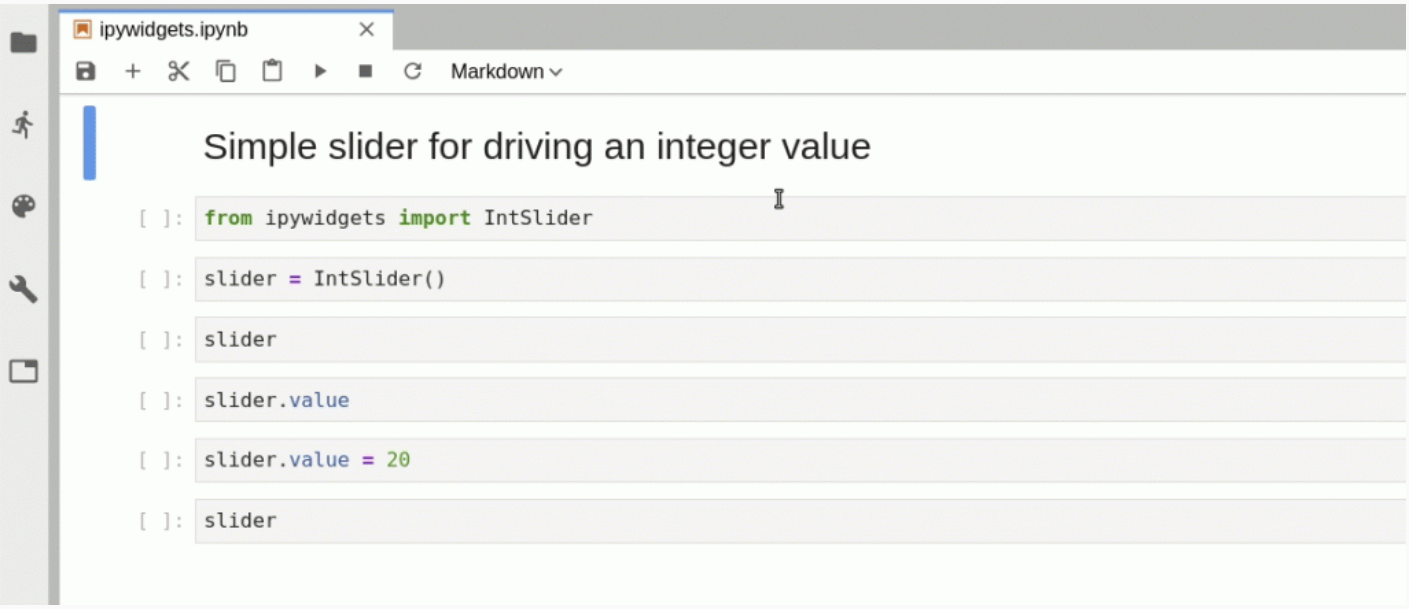
In this code we parse the page, fetch all the hyperlinks from the html, and get the text from those links.

```
Dr Ruth Abbott
Dr Gavin Alexander
Dr Edward Allen
Dr Scott Annett
Dr Rebecca Anne Barr
Prof Caroline Bassett
Dr Jenny Bavidge
Dr Joanna Bellis
Dr Jessica Berenbeim
Dr Kasia 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 step-by-step fashion through Jupyter Notebooks. They can also be used to create and share documents with live code, equations and visualizations.



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.