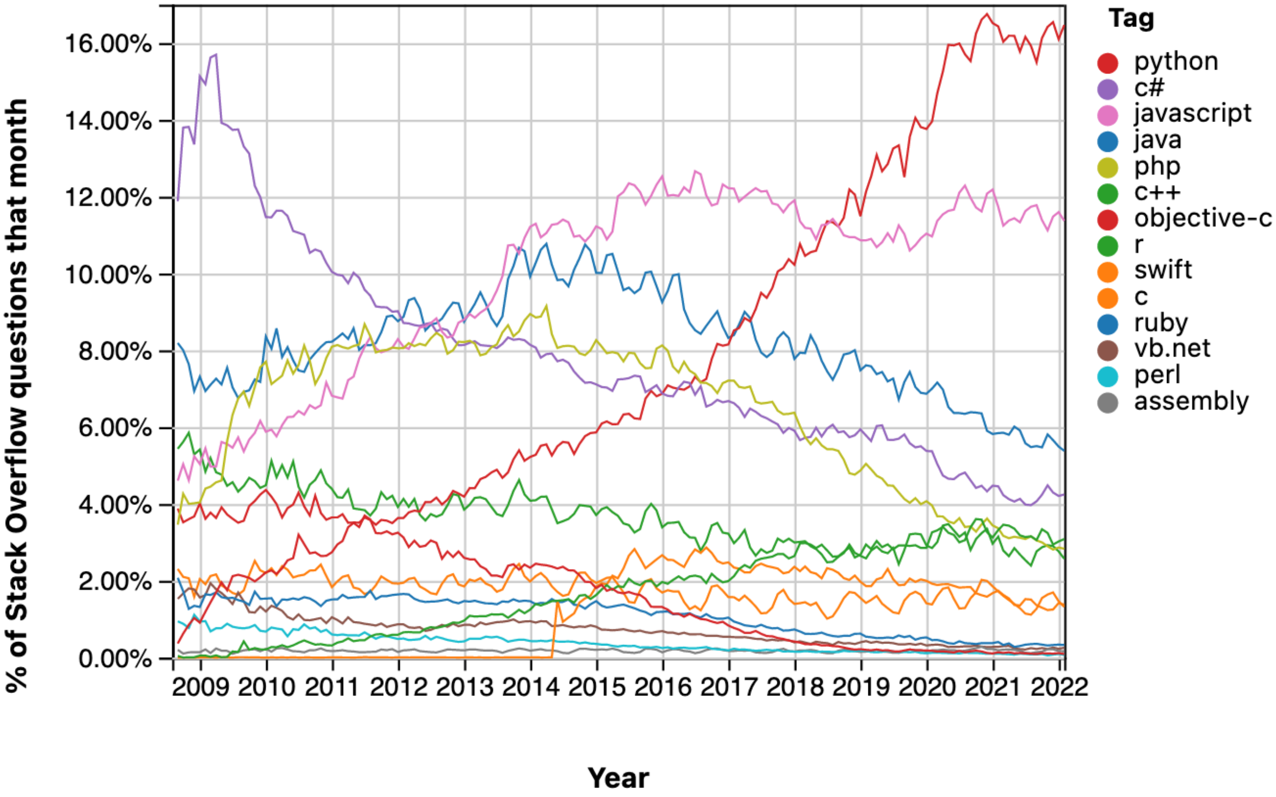
Introduction to Python

In this notebook, we will demo the basic functionality of Python programming language.

Why Python?

Python has become one of the most popular languages, particularly in the data and computational science. Below is an image from Stackoverflow (https://insights.stackoverflow.com/trends?tags=java%2Cc%2Cc%2B%2B%2Cpython%2Cc%23%2Cvb.net%2Cjavascript%2Cassembly%2Cphp%2Cc).



Here are some reason for pythons popularity:

1. Easy to learn and use
2. Supports various programming styles (OOP, functional, procedural)
3. Most of Python's utility comes from extensive collection of libraries/modules. For example, matplotlib is a library used for data visualizations. We will cover some of the most popular libraries soon.

Importing modules

```
In [1]: # Importing a module
        import numpy
```

```
In [2]: # accessing a modules parameters
        numpy.pi

Out[2]: 3.141592653589793
```

Comments

```
In [3]: # This is a single line comment
        """
        This is a
        multiline comment
        """

Out[3]: '\nThis is a\nmultiline comment\n'
```

Print Statements

```
In [4]: print("hello world")

        hello world
```

Python Data Types

Basic Variable Types

The basic data types in python are

- integers
- floats
- strings
- booleans
- complex

Python data types are dynamically inferred. Python distinguishes between integers and floats by whether the number contains a decimal place.

In [5]:

```
# Integer
x = 10
# Float
y = 10.
# String
z = "hello"
# Boolean
a = True

type(x),type(y),type(z),type(a)
```

Out[5]:

```
(int, float, str, bool)
```

Collections: List, Tuple, Dictionary

List

A list is a collection of objects (int, floats, string, any other python object):

- ordered
- changeable

In [6]:

```
example_list = [ "one", 2, 3. ]
```

List elements are indexed starting at zero. You can access elements as follows:

In [7]:

```
example_list[2]
```

Out[7]:

```
3.0
```

Lists are changeable, so we can do things like append a new value to the end of a list.

In [8]:

```
example_list.append("four")
example_list
```

Out[8]:

```
['one', 2, 3.0, 'four']
```

We can also compute the length of a list:

In [9]:

```
len(example_list)
```

Out[9]:

```
4
```

Tuple

Tuples are similar to list, but the are immutable. That is, tuples are

- ordered
- unchangable

```
In [10]: # Tuple
         example_tuple = (1, 2, 3)
```

```
In [11]: # Tuple : access element
         example_tuple[1]

Out[11]: 2
```

```
In [12]: # Get length of tuple
         len(example_tuple)

Out[12]: 3
```

```
In [13]: # Tuple : no append method i.e immutable
         #example_tuple.append(4)
```

Dictionary

Dictinaries store data as key:value pairs. You can access elements in a dictionary via the key.

```
In [14]: # Dictionary {Key:Value, Key:Value....}
         example_dictionary = {"a":1, "b":2, "c":3}
```

```
In [15]: # Dictionary : access element by key
         example_dictionary["c"]

Out[15]: 3
```

```
In [16]: # Add new element
         example_dictionary["d"]=4
         print(example_dictionary)

         # Get length of dictionary
         print(len(example_dictionary))

         {'a': 1, 'b': 2, 'c': 3, 'd': 4}
         4
```

Final Comments on Python Data Types

Everything in python is an object. That means that all data types have there on methods that manipulate the object. For example, append() is a method for the list data type. Here is another example.

```
In [17]: example_dictionary.keys()
Out[17]: dict_keys(['a', 'b', 'c', 'd'])
```

Control Flow

Before we demo python's control flow syntax, lets point out a few things about Python's syntax:

- Indentations matter in python; they specify code block boundaries

We will use the following list in the control flow demos:

```
In [18]: example_list = [1,2,3,4]
```

Before we demo the if/else syntax, below I highlight additional syntax included in this demo

```
In [19]: 3==0 # boolean operater == asks if elements are the same
Out[19]: False
```

```
In [20]: 4%2  # arithmetic: % is the remainder
Out[20]: 0
```

if/else statements

In the demo below, I determine if there are no elements in the list or if there are even or odd number of elements.

```
In [21]: if len(example_list)==0:  # boolean operater == asks if elements are the same
             print("empty list")
         elif len(example_list)%2==0:
             print("Even number of elements")
             print("!")
         else:
             print("Odd number of elements")
```

```
Even number of elements
!
```

For loop : Print even index elements of the list

In the demo below I print the even index elements of the list

In [22]:

```
for element in example_list:
    if example_list.index(element)%2==0:
        print(element)
    else:
        # pass does nothing
        pass
    print('test')
```

```
1
test
test
3
test
test
```

While loop : Search for "x" in list

In [23]:

```
example_list.append("x")
example_list
```

Out[23]:

```
[1, 2, 3, 4, 'x']
```

In [24]:

```
index=0
while index<len(example_list):
    if example_list[index] == "x":
        print("Found element x")
        break
    else:
        index+=1
        continue
    print('test')
```

```
Found element x
```

Defining Functions

Below we define a function to search a list for an element and return its index if found and -1 if not found.

In [25]:

```
def print_name(name):
    print('My name is {}'.format(name))

print_name('bob')
```

```
My name is bob
```

In [26]:

```
def search_list_for_element(element, search_list):
    index=0
    while index < len(search_list):
        if search_list[index] == element:
            break
        else:
            index+=1
            continue
    #If index is less than length, element is found
    if index < len(search_list):
        return index
    else:
        return -1
```

In [27]:

```
example_list
```

Out[27]:

```
[1, 2, 3, 4, 'x']
```

In [28]:

```
#search_list_for_element('z', example_list)
search_list_for_element('x', example_list)
```

Out[28]:

```
4
```

Lambda Functions

In [29]:

```
def y(x):
    return x**3
```

In [30]:

```
#Lambdas
y = lambda x:x**3
print(y(3))

z = (lambda x,y: x**y)
print(z(3,3))

#anonymous
print((lambda x,y: x**y)(3,3))
```

```
27
27
27
```

Exercise: Vector Dot Product

Write a function to compute a vector dot product of 2 lists:

- Define a function dot(x,y) that accepts 2 lists as arguments.
- Check if length of both lists are equal using len() function. If they are not equal print a message and return -1
- Generate the range of indices to iterate over
- Return the value of the dot product

Then, test it!

- Define 2 lists a=[1,2,3,4,5] and b=[6,7,8,9,10]
- Compute Dot Product with your function
- Print result

In [31]:

```
def dot(x,y):
    #check if both lists are equal length
    if len(x) == len(y):
        # empty list
        result_list = list()
        #index
        index = 0
        #iterate through 2 lists
        while index < len(x):
            result_list.append(x[index]*y[index])
            index+=1
        return sum(result_list)
    else:
        print("Error: Lists of unequal length given")
        return -1

a=list(range(1,6,1))

b=[6,7,8,9,10]

print(a, b, dot(a,b))
```

```
[1, 2, 3, 4, 5] [6, 7, 8, 9, 10] 130
```

In [ ]:

Optional: Basic I/O

Below we demo the syntax for input/output (I/O) operations.

In [32]:

```
file_path = 'data/8M_book.txt'
```

Below we open he file 8M_book.txt.

```
In [33]: file_object = open(file_path, "r")
```

```
In [34]: # Access file object attribute
         file_object.name
```

```
Out[34]: 'data/8M_book.txt'
```

```
In [35]: # read first 20 bytes
         first_n_bytes = file_object.read(20)
         first_n_bytes
```

```
Out[35]: '\ufeffThe Project Gutenbe'
```

```
In [36]: # tell file read pointer position
         print(file_object.tell())
```

```
22
```

```
In [37]: # seek back to zero
         file_object.seek(0)
```

```
Out[37]: 0
```

```
In [38]: #read a line
         line = file_object.readline()
```

```
In [39]: # tell file read pointer position
         print(file_object.tell())
```

```
67
```

```
In [40]: #file close
         file_object.close()
```

Line Count

```
In [41]: file_path = 'data/8M_book.txt'
```

In [42]:

```
# define a function to count number of lines in a file
#input: open file object, with seek position 0
#output: number of lines in the file
def count_number_of_lines(f):
    #read a line
    line = f.readline()
    #initiate line count
    if line:
        line_count=1
    else:
        line_count=0
    #iterate through each line of file
    while line:
        line_count+=1
        line = f.readline()
    return line_count

# Open a file
file_object = open(file_path, "r")

print(count_number_of_lines(file_object))

#file close
file_object.close()
```

```
146933
```

In [ ]:

In [ ]: