Python Basics – Data Structures

Contents

Contents

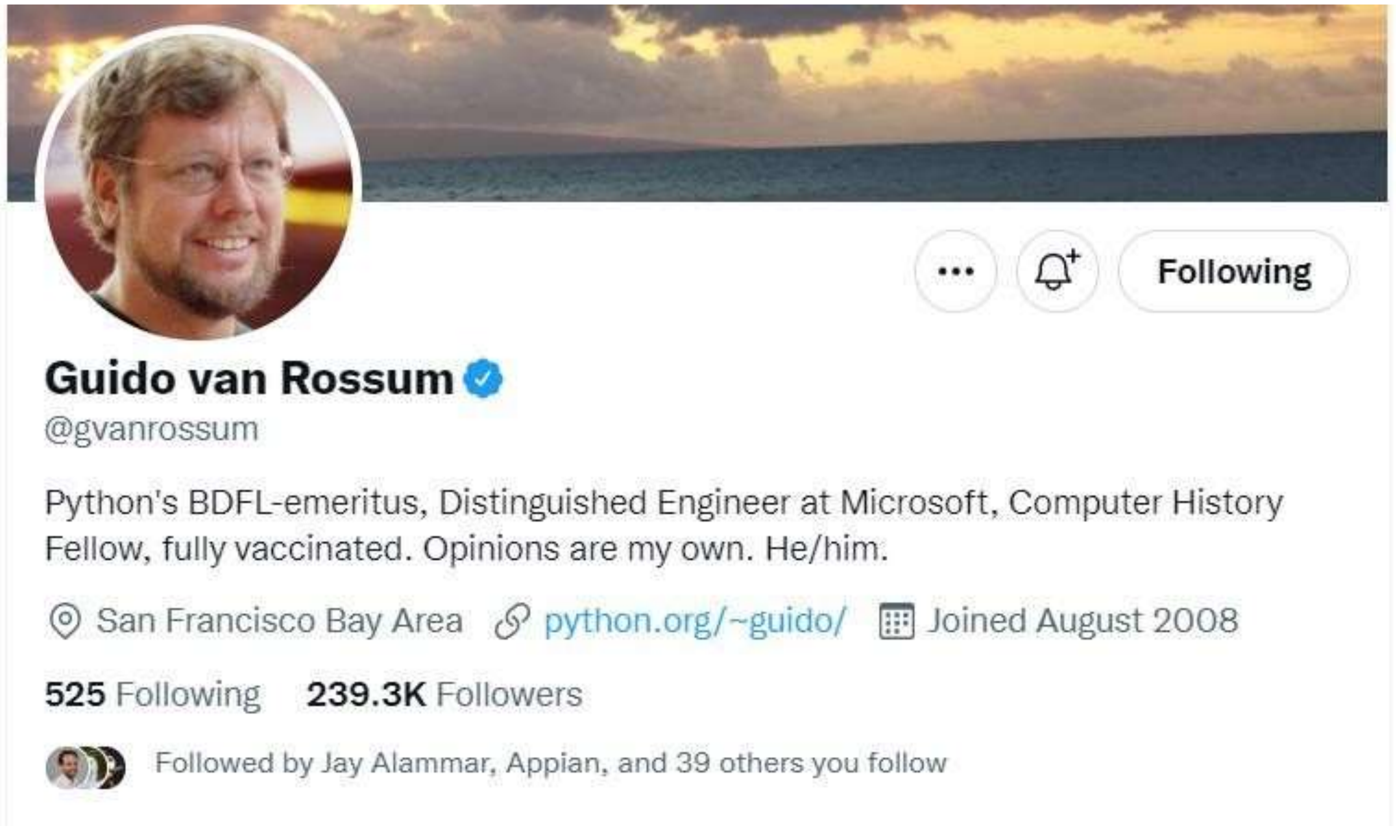
Introduction to Python & History

What is python

- It’s a multi purpose programming language
- Open Source (Free)
- Powerful scripting language with simple Syntax
- Used by many data scientists and developers
- Human-readable syntax and well Documented
- www.python.org

History

- Python language was developed by Guido van Rossum (Benevolent Dictator for Life).



Guido van Rossum

@gvanrossum

Python's BDFL-emeritus, Distinguished Engineer at Microsoft, Computer History Fellow, fully vaccinated. Opinions are my own. He/him.

San Francisco Bay Area python.org/~guido/ Joined August 2008

525 Following 239.3K Followers

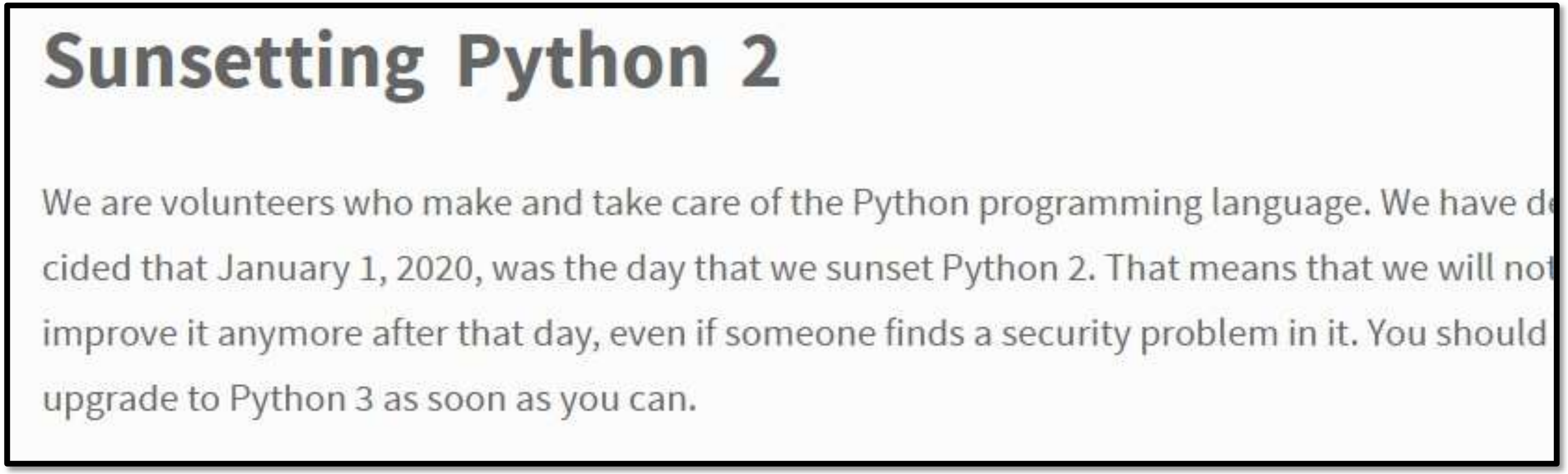
Followed by Jay Alammar, Appian, and 39 others you follow

Python History

- First Python version released in 1991
- Python 2 was released in 2000
- Python 3 was released in 2008
- Python 3 is **NOT** fully backwards compatible with Python 2

Python2 vs Python3

- Python 3 is **NOT** fully backwards compatible with Python 2



Sunsetting Python 2

We are volunteers who make and take care of the Python programming language. We have de
cided 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.

Installing Python & Python IDEs

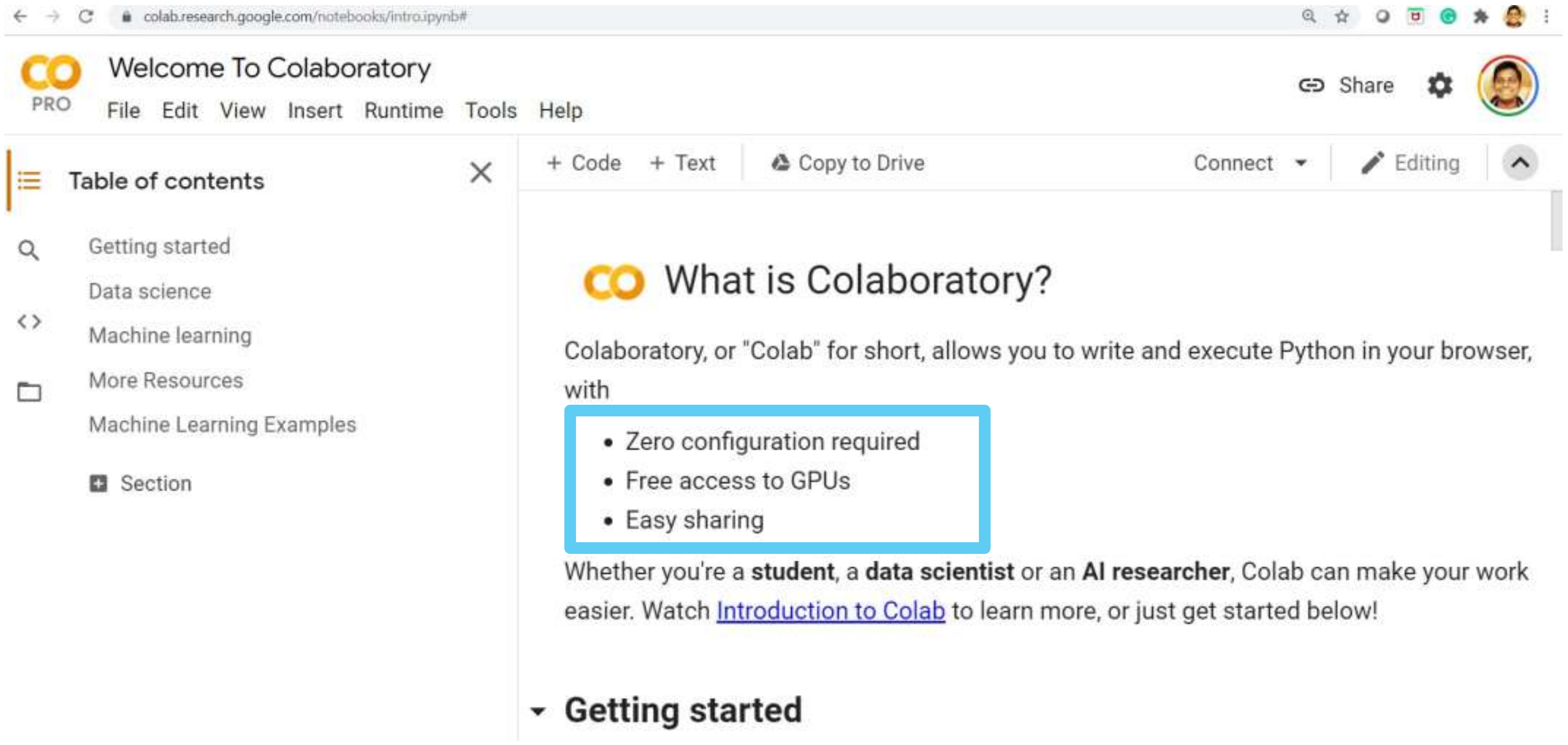
Writing and executing python programs

- Python has many options to write and execute a program
- You can use Text Editors or Command line interfaces or Notebook or an IDE
- Anaconda distribution has all the required software's inbuilt. We just need to download and install it.

Installing Python, Anaconda

- Download and install Anaconda3
- It automatically installs
  - Ipython
  - Jupyter notebook
  - Spyder IDE

Python on Cloud – Google Colab



What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

Getting started

Python Version

```
from platform import python_version
print(python_version())
```

Basic Commands in Python

Before you code

- Python is case sensitive
- Be careful while using the Variable names and Function names
  - `Sales_data` is not same as `sales_data`
  - `Print()` is not same as `print()`

Basic Commands

```
571+95
19*17
print(57+39)
print(19*17)
print(“Statinfer")

#use hash(#) for comments
#Division example
34/56
```

Basics-What an error looks like?

```
Print(600+900) #used Print() instead of print()
576-'96'
```

Assigning and Naming convention

Assignment operator

= is the assignment operator

```
income=12000
income

x=20
x

y=30
z=x*y
z

name="Jack"
name
print(name)
```

```
del x #deletes the variable
```

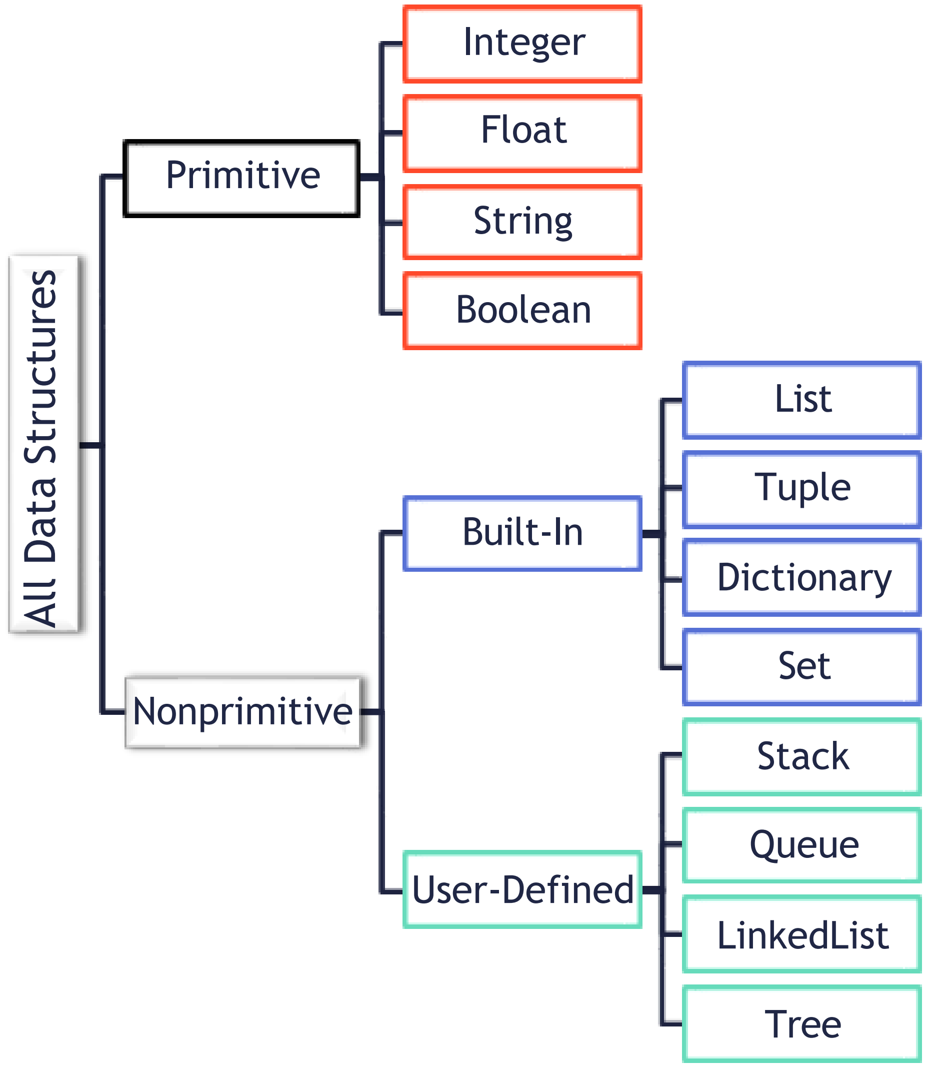
Naming convention

- Must start with a letter (A-Z or a-z)
- Can contain letters, digits (0-9), and/or underscore "_"

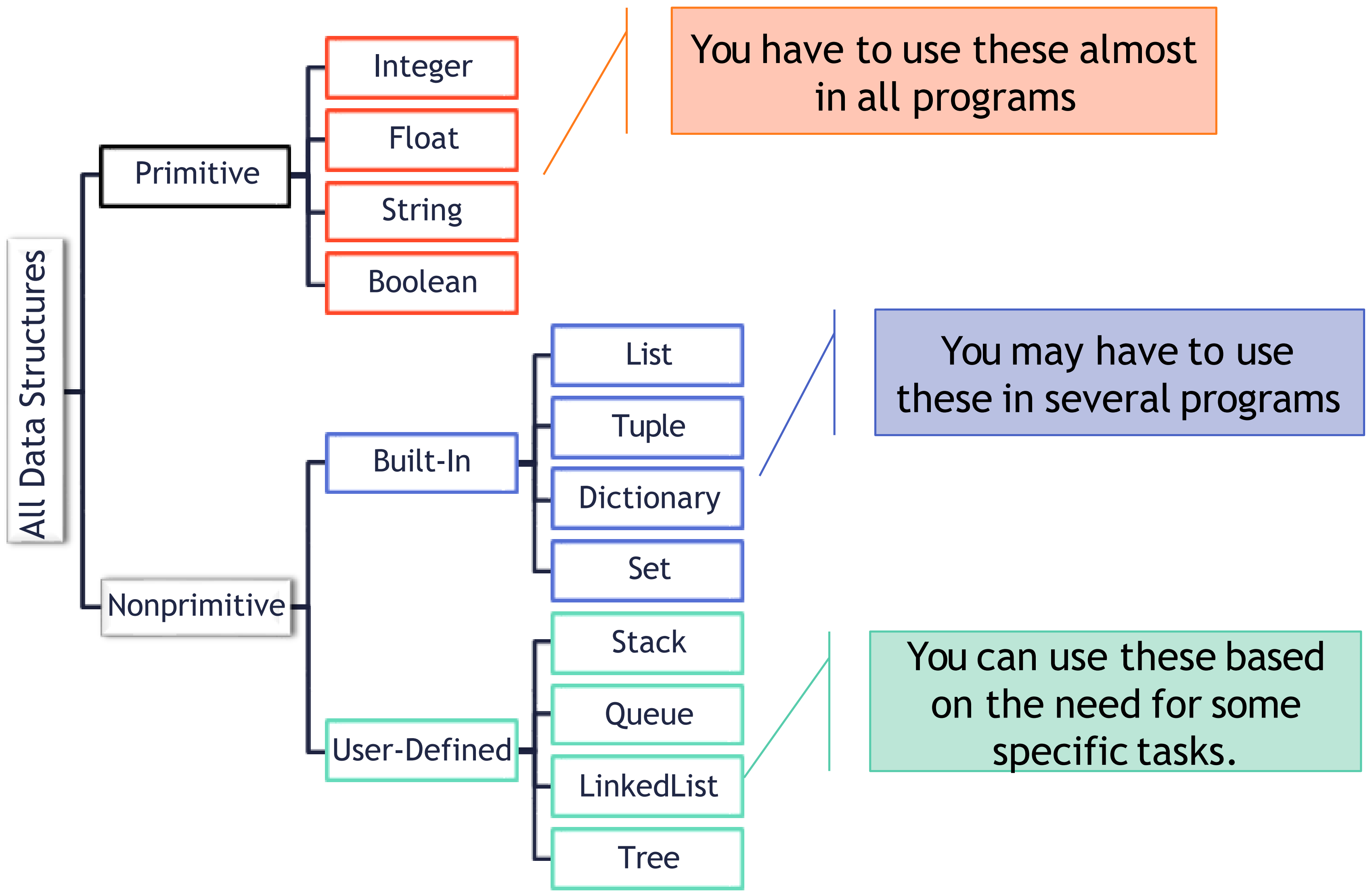
```
1x=20 #Doesn't work

x1=20 #works
x1

x.1=20 #Doesn't work
x.1

x_1=20 #works
x_1
```

Python Data Structures

Python Data Structures



Python Data Structures


All Data Structures
Primitive
Integer
Float
String
Boolean
Nonprimitive
Built-In
List
Tuple
Dictionary
Set
User-Defined
Stack
Queue
LinkedList
Tree
You have to use these almost in all programs
You may have to use these in several programs
You can use these based on the need for some specific tasks.

Integers and Float

- Numbers: integers & floats

```
age=30
income=1250.567

print(type(age))
print(type(income))

print("Age - datatype", type(age))
print("Income - datatype", type(income))
```

Strings

- Strings are amongst the most popular types in Python.
- There are a number of methods or built-in string functions

Defining Strings

```
name="Sheldon"
print(name)
```

Accessing strings

```
print(name[0])
print(name[0:3])
print(name[3:6])
print(name[3:4])
```

Strings – Slicing

```
message="Python 5 day course - Statinfer"
print(message)
print(message[0:4])
print(len(message))
```

String Concatenation

```
First_name="Sheldon"
Last_name="Cooper"
Name= First_name + Last_name

Name1= First_name +" "+ Last_name
print(Name1)
print(Name1*10)
```

LAB : Strings

- Create a string “ Hello World ”. Beware of the empty spaces before and after.
- What is the index position of the ‘r’?
- Remove the first and last space from it.
- Extract “ello” from it.
- Access ‘W’ using it’s index location.
- Add “!” at the end.

Boolean

- Used to store and return the values True an False.
- Useful for comparison operations

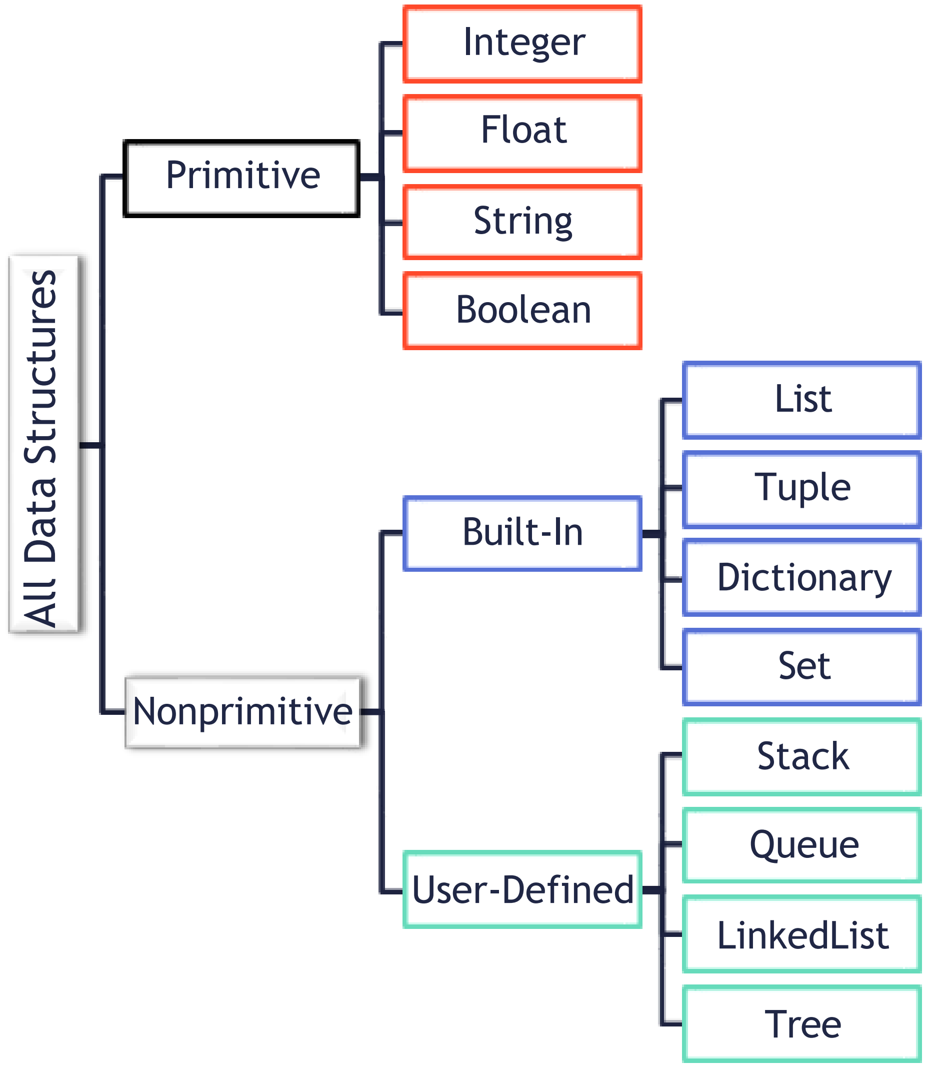
```
a=5>6
print(a)
print(type(a))
```

Boolean

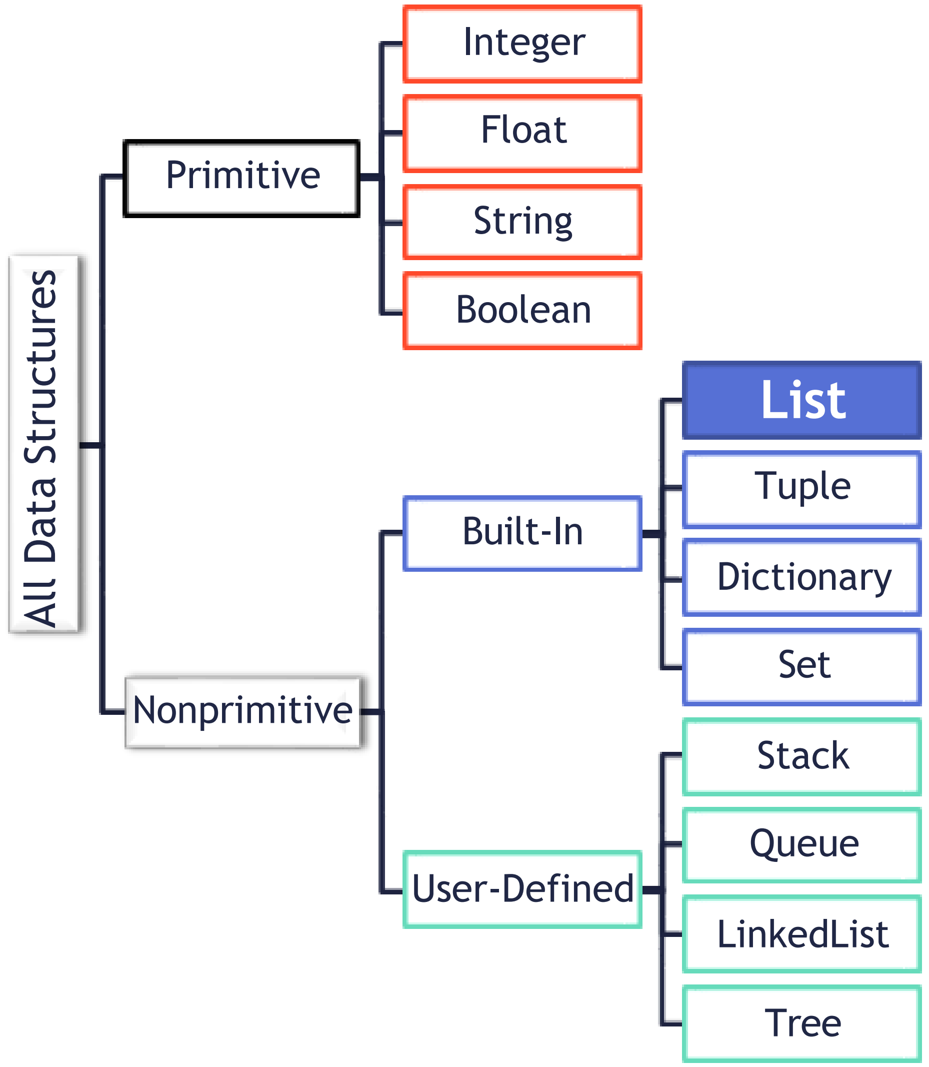
What is the expected output of this code

```
a=5>6
if a:
  print(10)
else:
  print(15)
```

Python Data Structures



Python Data Structures



List

- A sequence or a collection of elements.
- A list looks similar to an array, but not array
- Lists, can contain any number of elements. The elements of a list need not be of the same type

List Creation

Creating a list

```
cust=["Sheldon","leonard", "penny"]
Age=[30, 31, 27]

print(cust)
print(Age)
```

Accessing list elements

```
print(cust[0])
print(Age[1])
```

Combining two lists

```
cust_1=["Amy", "Raj"]
Final_cust=cust + cust_1
print(Final_cust)
print(len(Final_cust))
```

Appending an item to the list

```
Final_cust.append("Howard")
print(Final_cust)
```

Delete – based on value (a.k.a – remove)

- The remove() method removes the first matching element (which is passed as an argument) from the list.

```
Final_cust.remove("penny")
print(Final_cust)
```

- What is the expected result for the below code?

```
Age=[30, 31, 27, 30, 48]
Age.remove(30)
print("After removing 30 ==>", Age)
```

Delete – based on index (a.k.a – pop)

- The pop() method removes the item at the given index from the list and returns the removed item.

```
sales=[300, 350, 200, 250, 300]
sales.pop(0)
print("sales after pop ==>", sales)
```

What is the expected result for the below code?

```
sales.pop(0)
print("sales after second pop ==>", sales)
```

Sort the list items

```
Loans=[3, 0, 2, 6, 20]
Loans.sort()
print(Loans)
```

```
Loans.sort(reverse=True)
 print(Loans)
```

Negative Index

List	300	350	200	250	400
Usual Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

```
sales=[300, 350, 200, 250, 400]
print(sales[-1])

print(sales[-1],sales[-2], sales[-3], sales[-4], sales[-5])
```

Slicing a list

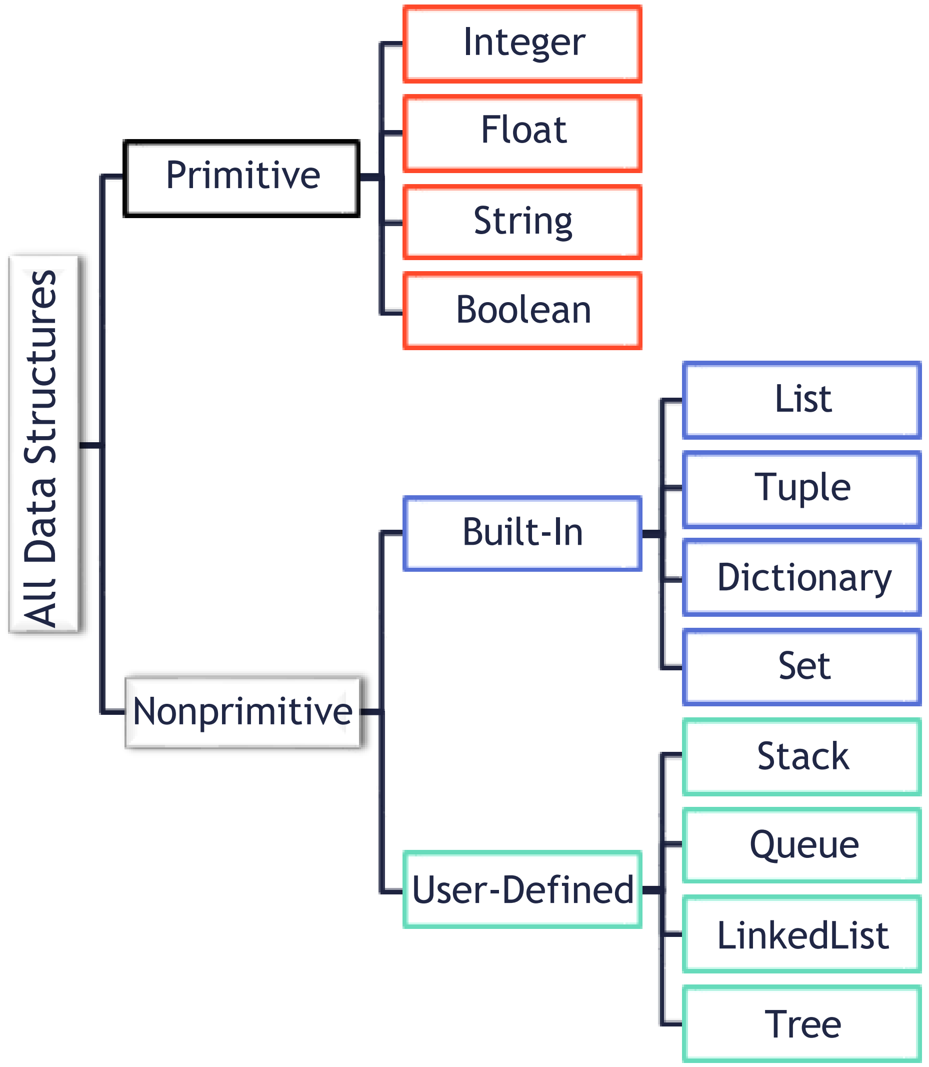
What is the expected result for the below code?

```
Age=[15, 44, 38, 19, 36, 25]
print(Age[2:5])
print(Age[2:10])
print(Age[2:])
print(Age[:4])
print(Age[-4:-2])
print(Age[:-4])
print(Age[-4:])
```

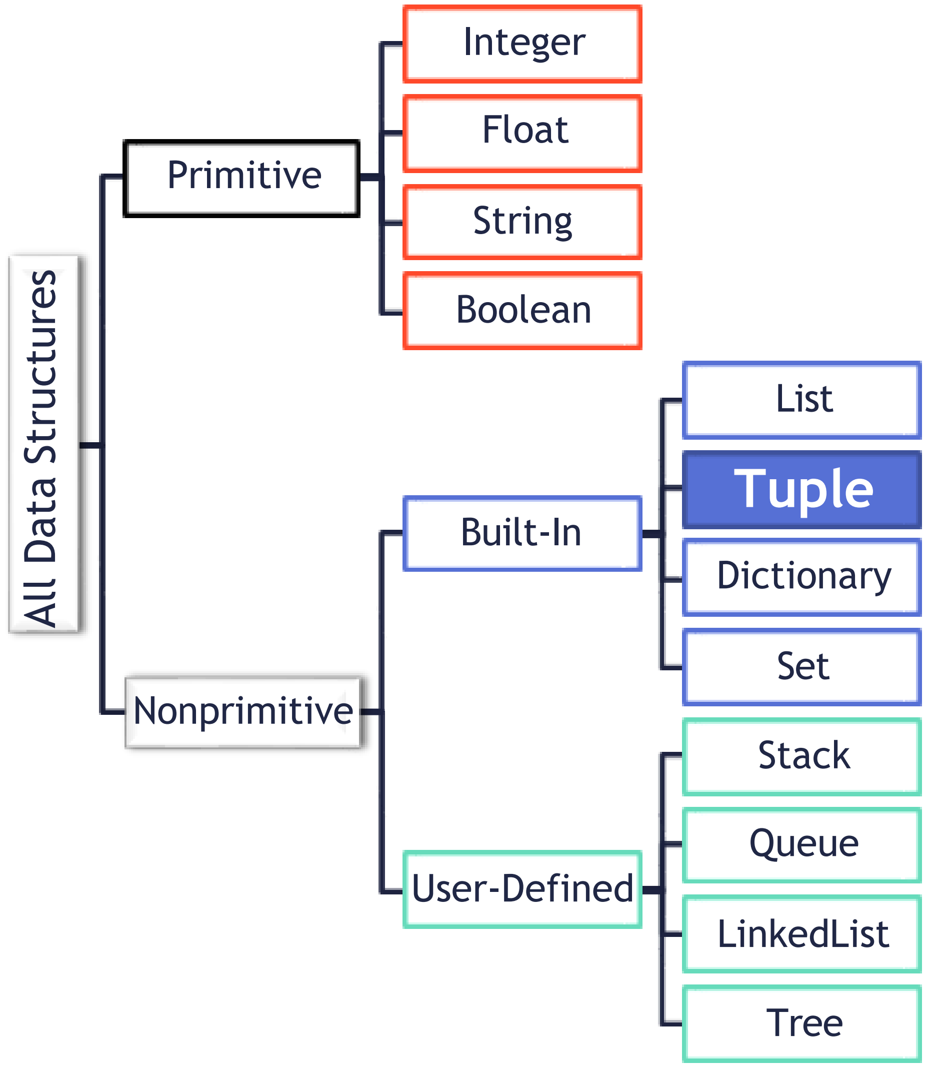
LAB: Lists

- Create a list with these numbers:
  - `names = ['Stan', 'Kyle', 'Cartman', 'Kenny']`
- Change the value of ‘Kenny’ to ‘Butters’.
- Insert ‘Clyde’ at second position, after Stan ? (hint use insert())
- Remove butters but store the list in a variable called **b.**
- Access last 3 element of this list using negative indexing
- Define a list Age=[15, 44, 38, 19, 36, 25]
  - How to reverse the items in a list. (hint use reverse())
  - How to access the minimum and maximum values of a list?
  - How to know the count of all elements inside a list ?
  - How to access the index of a particular value inside a list?

Python Data Structures



Python Data Structures



Tuples

- Tuple is one of 4 built-in data types in Python
- Collection of items stored in a single variable
- Items inside a tuple can be homogeneous or heterogenous.
- Tuples are sequences, just like lists. There are some major differences between tuples and lists.
- Tuples are immutable. We can NOT update or change the values of tuple elements

Tuples – Defining and Accessing

```
custd_id=("c00194", "c00195", "c00198")
type(custd_id)

rank=(1, 46, "NA", 5, 8)
type(rank[1])
print(type(rank[1]),type(rank[2]))
```

Tuple packing

```
region= "E", "W", "N", "S"
type(region)
```

- The above operation is called tuple packing. "E", "W", "N", "S" are packed together.
- The reverse is also possible.

```
r1, r2, r3, r4= region
print(r1, r4)
print(type(r1))
```

Tuples vs. Lists – Difference

- Tuples are immutable. Seal the data in the tuple, totally prevent it from being modified at any stage of the development.
- Tuples are faster and use less memory than lists.

```
custd_id_t=("c00194", "c00195", "c00198")
custd_id_l=["c00194", "c00195", "c00198"]
custd_id_l[1]="c00176"
custd_id_t[1]="c00176"
```

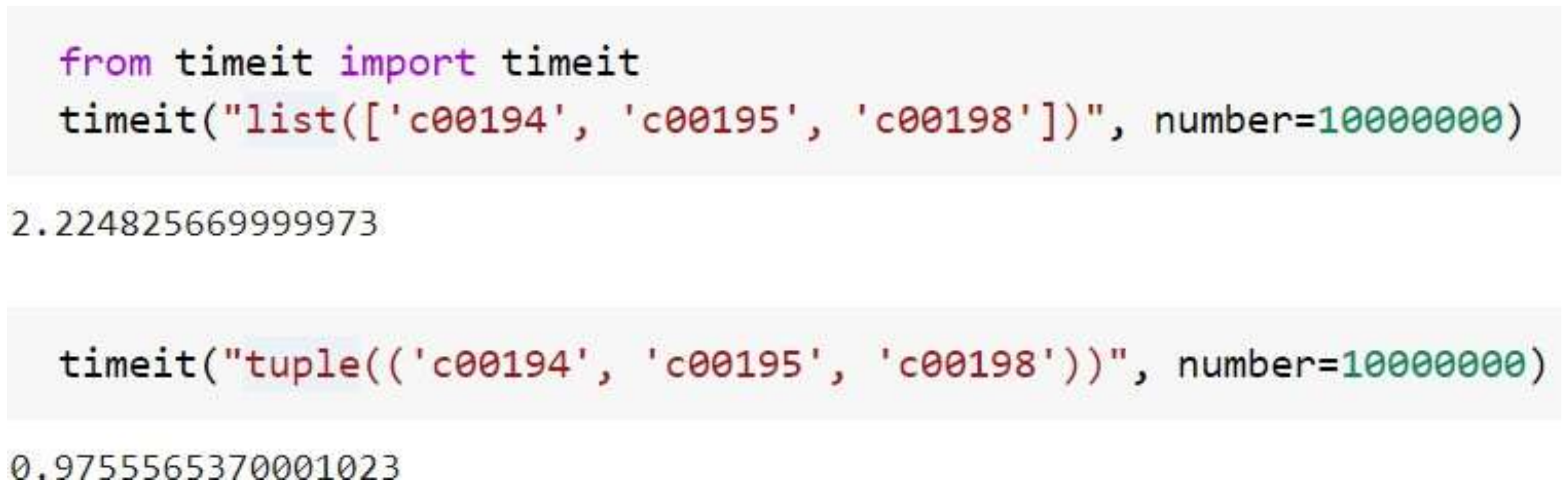
Tuples utilize less memory

- A list is mutable. Python needs to allocate more memory than needed to the list. This is called over-allocating.
- Meanwhile, a tuple is immutable. When the system knows that there will be no modifications, memory allocations will be very efficient.

```
from sys import getsizeof

custd_id_t=("c00194", "c00195", "c00198")
custd_id_l=["c00194", "c00195", "c00198"]

print("tupple size ", getsizeof(custd_id_t))
print("list size ", getsizeof(custd_id_l))
```

Working with a tuple is faster

- Time that needs to copy a list and a tuple 1 billion times



```
from timeit import timeit
timeit("list(['c00194', 'c00195', 'c00198'])", number=10000000)
```

```
2.224825669999973
```

```
timeit("tuple(('c00194', 'c00195', 'c00198'))", number=10000000)
```

```
0.975565370001023
```

Tuple vs List – Which one to use?

- We have to use both.
- Sometimes we need mutability and modifications to the variables, in some other cases, we need speed and utilize less memory.

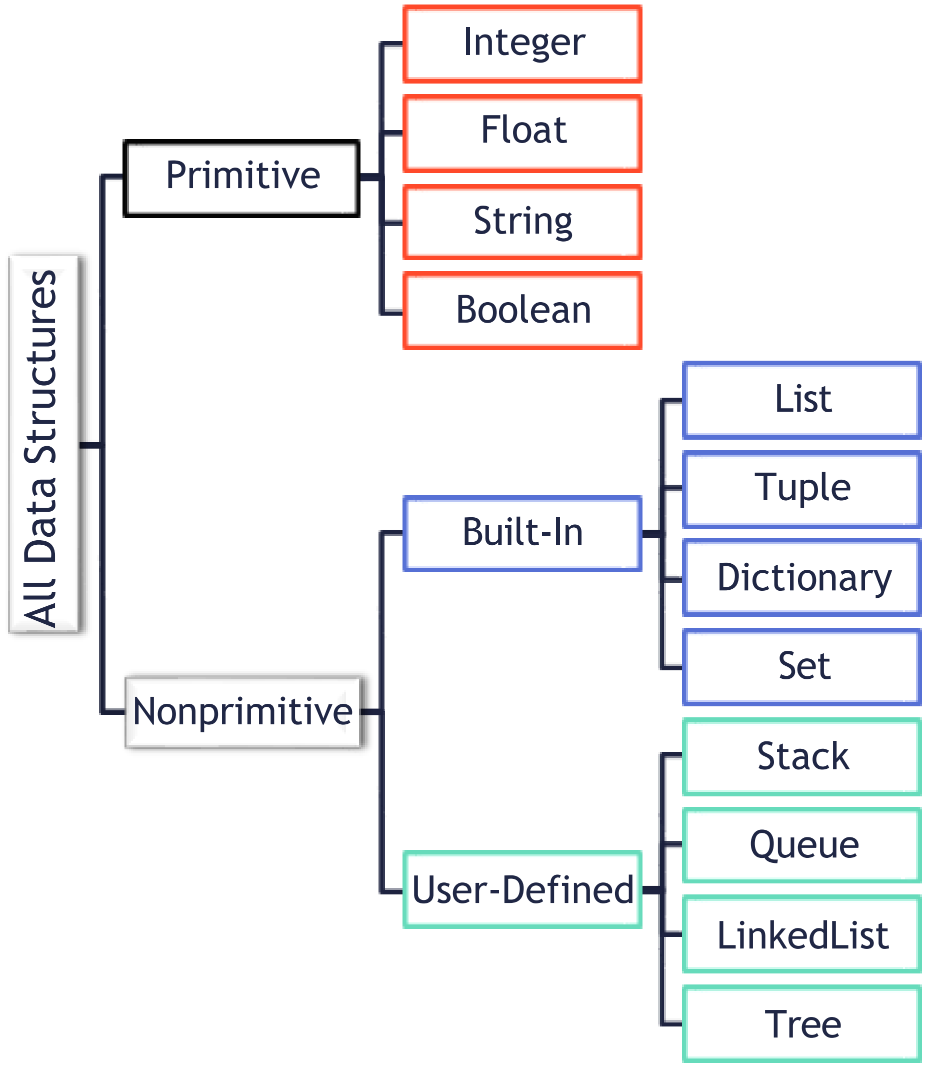
Tuple vs List – Which one to use?

- Example-1
  - You are writing a function. You need to take the input parameters from the user and perform the calculations inside the function to return some result.
  - How would you like to store the input parameters ? Inside a tuple? Or inside a list?
- Example-2
  - Store all the columns inside a variable and return the first two columns. Later, rename the first column and pass it on to the next step.
  - How would you like to store the column names? Inside a tuple? Or inside a list?

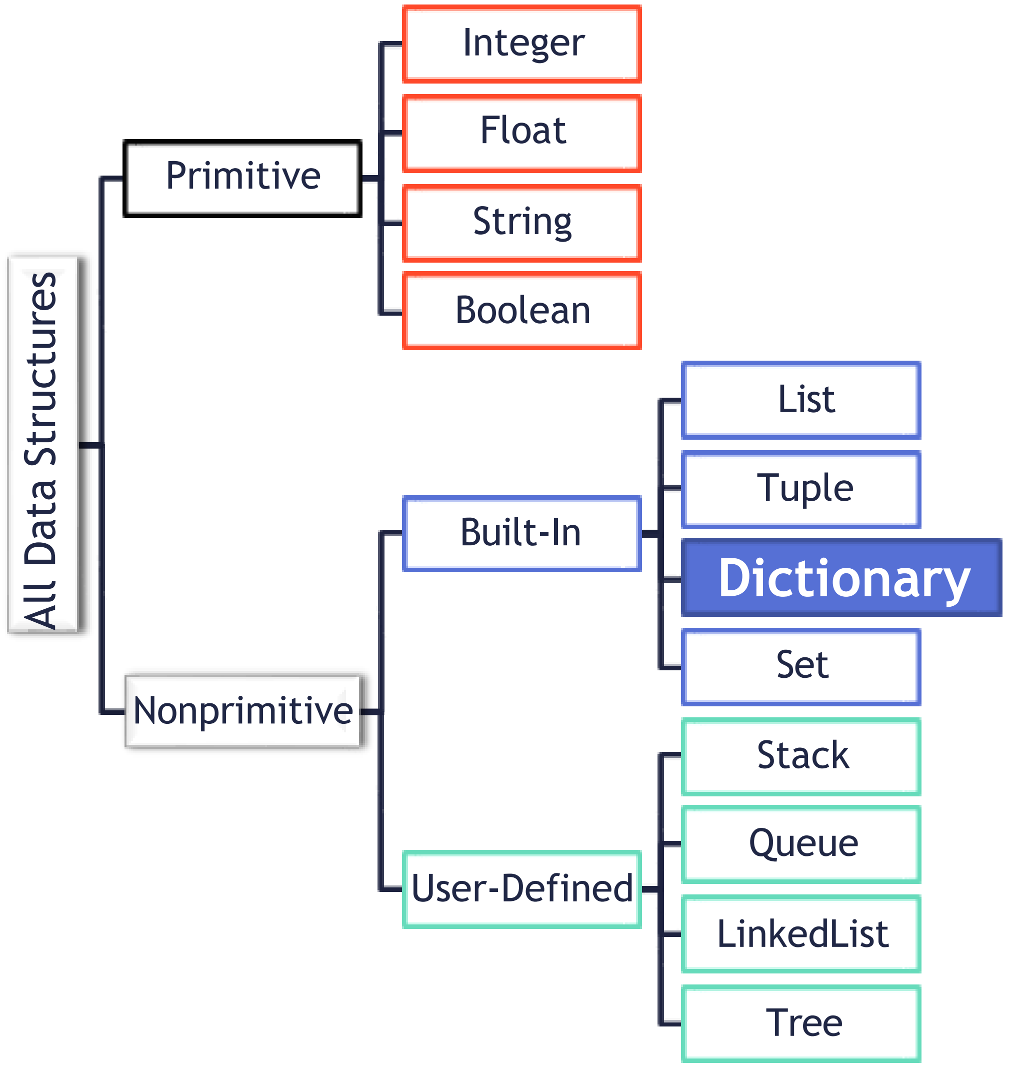
LAB: Tuple

- Create a tuple:
  - (28, ‘Sophia’, True, names) | beware the names is a list containing 4 names
- Access first element of the names from this tuple.
- Re-assign a new value for the last element of the list names via this tuple.
- Print the list names and see if it has changed.

Python Data Structures



Python Data Structures



Dictionaries

- Dictionaries have two major element types key and Value.
- Dictionaries are collection of key value pairs
- Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces.
- Keys are unique within a dictionary

```
city={0:"L.A", 9:"P.A", 7:"FL"}
type(city)

print(city)
print(city[1])
print(city[0])
```

Dictionaries

- In dictionary, keys are similar to indexes. We define our own preferred indexes in dictionaries

```
cust_profile={"C001":"David", "C002":"Bill", "C003":"Jim"}
print(cust_profile[0])
print(cust_profile[C001])
print(cust_profile["C001"])

#Updating values
cust_profile["C002"]="Tom"
print(cust_profile)
```

Dictionaries

Updating keys in dictionary

Delete the key and value element first and then add new element

```
#Updating Keys
#C001 ---> C009

#Delete + Add

del(cust_profile["C001"])
print(cust_profile)

cust_profile["C009"]="David"
print(cust_profile)
```

Dictionaries

- Fetch all keys and all values separately

```
city.keys()
city.values()
```

Iterating in a Dictionary

```
Employee = {"Name": "Tom", "Emp_id": 198876, "Age": 29, "salary":25000
,"Company":"FB"}
```

- What is the expected output?

```
for x in Employee:
    print(x)
```

- What is the expected output?

```
for x in Employee.values():
    print(x)
```

Iterating in a Dictionary

- What is the expected output?

```
for x in Employee.keys():
    print(x,Employee[x])
```

- You can also use items.

```
for x in Employee.items():
    print(x)
```

LAB: Dictionary

- Create a the below dictionary with 3 names as keys and their age as values.
  - `name_age = {'Stan':8, 'Kyle':8, 'Cartman':10}`
- Access the age of 3rd key.
- Append dictionary with single key and value pair. Kenny - 9
- Update the dictionary with a two values `{'Cartman':11, 'Token':10}`