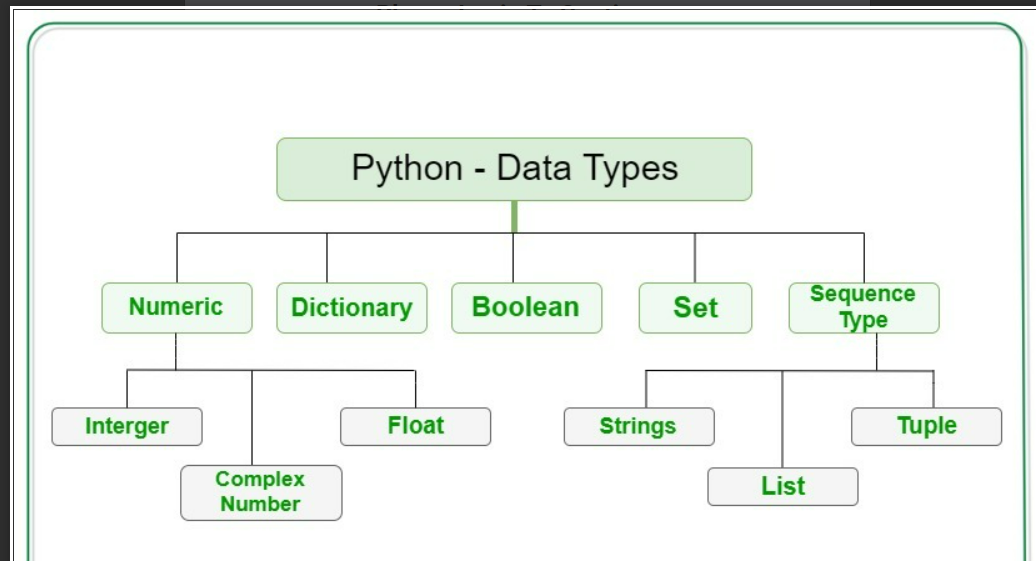
PYTHON DATATYPES



# NUMBERS

• int  
• float  
• complex

x = 1 # int  
y = 2.8 # float  
z = 1j # complex

# STRING

# Python Program for  
# Creation of String  
  
# Creating a String  
# with single Quotes  
String1 = 'Welcome to the Automation World'  
print("String with the use of Single Quotes: ")  
print(String1)  
  
# Creating a String  
# with double Quotes  
String1 = "I'm a QA"  
print("\nString with the use of Double Quotes: ")  
print(String1)  
print(type(String1))  
  
# Creating a String  
# with triple Quotes  
String1 = '''This is how a Automation code should work"'''  
print("\nString with the use of Triple Quotes: ")  
print(String1)  
print(type(String1))

## Slicing and Indexing

“Indexing” means referring to an element of an iterable by its position within the iterable. “Slicing” means getting a subset of elements from an iterable based on their indices.

**Example: 1**

**Str -- > Automation is important**

Indexing:   
Automation is important  
0123456789...........20 (From Left to right)  
-21,-20,-19...........-1 (From Right to Left)

Example 2:

0 1 2 3 4 5 6 7 8  
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓  
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']  
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑  
-9 -8 -7 -6 -5 -4 -3 -2 -1

print(String1.lower())

python automation

print(String1.upper())

PYTHON AUTOMATION

print(String1.replace("Python","Advanced"))

Advanced Automation

name= " Python "  
print(name.strip())

Python

str1= "automation"  
print(str1.capitalize())

Automation

## Slicing

* Syntax for slicing is [start : stop : step]
* start is the starting index from where to slice a list or tuple
* stop is the ending index or where to sop.
* step is the number of steps to jump.
* Default value for start is 0, stop is number of items, step is 1.
* Slicing can be done on strings, arrays, lists, and tuples.

#Slicing  
print(str[0:13])  
print(str[:13])  
print(str[5:13])  
print(str[5:])

## Stepping

Example:

0 1 2 3 4 5 6 7 8  
↓ ×¹ ↓² ×¹ ↓² ×¹ ↓² ×¹ ↓²  
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

my\_list=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']  
print(my\_list[1::2])

['b', 'd', 'f', 'h']

# LISTS

* Ordered
* Mutable
* Allow Duplicates

# a list of programming languages  
languages=['Python', 'C++', 'JavaScript']  
  
# empty list  
my\_list = []  
  
# list with mixed data types  
my\_list = [1, "Hello", 3.4]  
  
  
list1 = ['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']  
  
# first item  
print(list1[0])

apple  
  
# third item  
print(list1[2])

banana  
  
# fifth item  
print(list1[4])

grape  
  
# Nested List  
n\_list = ["Happy", [2, 0, 1, 5]]  
  
# Nested indexing  
print(n\_list[0][1])  
a

print(n\_list[1][3])

5

## Accesing,changing and adding List

### Insert,Append,Remove,Pop

#Change the list item based on index  
languages[0]= 'pearl'  
print(languages)  
['pearl', 'C++', 'JavaScript']

#Add item to list based on position  
languages.insert(2, "Python")  
print(languages)

['pearl', 'C++', 'Python', 'JavaScript']

#Add item to the end of the list  
languages.append("Python")  
print(languages)

['pearl', 'C++', 'Python', 'JavaScript', 'Python']

#Add a new list to existing list  
languages.extend(["C++","Ruby"])  
print(languages)

['pearl', 'C++', 'Python', 'JavaScript', 'Python', 'C++', 'Ruby']

#Remove Specified Item  
languages.remove("JavaScript")

print(languages)

['pearl', 'C++', 'Python', 'Python', 'C++', 'Ruby']

#Remove the second item  
languages.pop(1)

print(languages)

['pearl', 'Python', 'Python', 'C++', 'Ruby']

### Split and Join

* **split()** function to split a string based on a delimiter to a list of strings.
* **join()** function to join a list of strings based on a delimiter to give a single string.

string = "This is a string."  
string\_list = string.split(' ')

print(string\_list)

#output: ['This', 'is', 'a', 'string.']  
print(' '.join(string\_list))

#output: This is a string.

# TUPLES

# Creating an empty tuple  
Tuple1 = ()  
print(Tuple1)  
('Cannot', 'change', 'tuple')

# Creating a Tuple with  
# the use of Strings  
Tuple1 = ('Cannot','change','tuple')  
print(Tuple1)

(1, 2, 4, 5, 6)

# Creating a Tuple with the use of list  
list1 = [1, 2, 4, 5, 6]  
print(tuple(list1))

# Creating a Tuple with nested tuples  
Tuple1 = (0, 1, 2, 3)  
Tuple2 = ('python', 'automation')  
Tuple3 = (Tuple1, Tuple2)  
print(Tuple3)  
((0, 1, 2, 3), ('python', 'automation'))

Tuple3[0]='not possible'

TypeError: 'tuple' object does not support item assignment

#count the item

print(Tuple2.count('python'))

1

#give position of the item

print(Tuple2.index('python'))

2

# Tuples vs Lists

**Lists**and **Tuples** are both s**equence data types** that can store a collection of objects in Python. The objects stored in both sequences can have **different data types**. Lists are represented with **square brackets** ['sara', 6, 0.19], while tuples are represented with **parantheses** ('ansh', 5, 0.97).

while tuples are immutable objects, lists are mutable. This means tuples cannot be changed while lists can be modified. Tuples are also more memory efficient than the lists.

# Dictionary

* Ordered
* Mutable
* **No** Duplicates

Dictionary is faster than Lists

Efficient to use dictionaries for the lookup of elements as it is faster than a list and takes less time to traverse.

## Items,values,keys

first\_dict = {  
 "name": "Employee1",  
 "Dept": "QA",  
 "type": "Automation",  
 "age": 28,  
 "price": "free",  
 "work-style": "remote",  
}  
  
  
print(first\_dict['Dept'])

QA  
  
print(first\_dict.get('Dept'))

QA  
  
print(first\_dict.get('Company','XYZ'))

XYZ  
  
dict\_keys = first\_dict.keys()  
print(dict\_keys)

dict\_keys(['name', 'Dept', 'type', 'age', 'price', 'work-style'])  
  
dict\_values = first\_dict.values()  
print(dict\_values)

dict\_values(['Employee1', 'QA', 'Automation', 28, 'free', 'remote'])

items = first\_dict.items()  
print(items)

dict\_items([('name', 'Employee1'), ('Dept', 'QA'), ('type', 'Automation'), ('age', 28), ('price', 'free'), ('work-style', 'remote')])  
  
  
first\_dict.pop("work-style")  
print(first\_dict)

{'name': 'Employee1', 'Dept': 'QA', 'type': 'Automation', 'age': 28, 'price': 'free'}

first\_dict.popitem()  
print(first\_dict)

{'name': 'Employee1', 'Dept': 'QA', 'type': 'Automation', 'age': 28}

## Dict: Nested List and dicts

#Nested dictionary with a list and dictionary  
main\_dict = {  
 "name": "Employee1",  
 "Dept": "QA",  
 "type": ["Automation","Manual"],  
 "age": 28,  
 "price": {"Morning": 100,"Night" : 200},  
 "work-style": "remote"  
}  
  
print(main\_dict["price"]["Morning"])

100

print(main\_dict["type"][1])

Manual

main\_dict.clear()  
print(main\_dict)

{}

# SETS

set ={}  
  
#Create a list with unique values  
lang = {'Python', 'C++', 'Java', 'C'}  
lang.add("Ruby")  
print(lang)

{'Java', 'C++', 'Python', 'Ruby', 'C'}

#Duplicates are not allowed  
lang.add("Ruby")  
print(lang)

{'Java', 'C++', 'Python', 'Ruby', 'C'}  
  
#Intersection set  
nums = {1, 2, 3, 4, 5 }  
oddNums = {1, 3, 5, 7, 9}  
nums.intersection\_update(oddNums)  
print("Updated set: ", nums)

Updated set: {1, 3, 5}

nums1 = {1, 2, 2, 3, 4, 5}  
nums2 = {4, 5, 6, 7, 7, 8}  
distinct\_nums = nums1.union(nums2)  
print("The union of two sets is: ", distinct\_nums)

The union of two sets is: {1, 2, 3, 4, 5, 6, 7, 8}

# Builtin Methods:

## ZIP,ALL,Type,Sorted

#ZIP  
  
numbers = [1,2,3]  
str\_numbers = ['One','Two','Three']  
result = zip(numbers, str\_numbers)  
print(result)  
print(list(result))  
  
#ALL  
lst = []  
print(all(lst)) # Returns True for empty list  
  
nums = [1, 2, 3, -4, -5]  
print(all(nums)) # Returns True  
  
nums = [0, 1, 2, 3]  
print(all(numbers)) # Returns False because of 0  
  
data = [1, 'Phone', 12.5, 5, False]  
print(all(data)) # Returns False because of False  
  
#Type  
lang = 'Python'  
nums = [1,2,3,4]  
nums\_dict = {'one':1,'two':2,'three':3}  
  
print(type(nums))  
print(type(lang))  
print(type(nums\_dict)

<class 'list'>

<class 'str'>

<class 'dict'>

nums = [2,1,5,3,4]  
asc\_nums = sorted(nums)  
dsc\_nums = sorted(nums, reverse = True)  
print("Ascending Numbers: ", asc\_nums)  
print("Descending Numbers: ", dsc\_nums)

Ascending Numbers: [1, 2, 3, 4, 5]

Descending Numbers: [5, 4, 3, 2, 1]

BOOLEANS

print(10 > 9)

True

print(10 == 9)

False

print(10 < 9)

False