**Python Sequence**

It is a type that stores more than one item at a time. Therefore, we can say that it is a collection of items. Moreover, to access these items each item has a particular index number. There are three types of sequence data types namely, strings, lists, and tuples.sequence data types represent a sequence or ordered collection of characters. This means that you can access individual characters in a string by their index, iterate over the characters using loops, and perform various sequence operations such as slicing and concatenation.

**String:**

Python does not have a character data type, a single character is simply a string with a length of 1.

Since strings are arrays, we can loop through the characters in a string.

To get the length of a string, use the len() function.

**Slicing**

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

[a:b:c]

The split() method splits the string into substrings if it finds instances of the separator:

a = "a,b,c,d,e"

print(a.split(",")) # ['a', 'b', 'c', 'd', 'e']

a = "Hello, World!"

print(a.replace("H", "J"))

**F-Strings**

F-String was introduced in Python 3.6, and is now the preferred way of formatting strings.

To specify a string as an f-string, simply put an f in front of the string literal, and add curly brackets {} as placeholders for variables and other operations.

price = 59

txt = f"The price is {price:.2f} dollars"

print(txt)

**Escape Character**

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

txt = "We are the so-called \"Vikings\" from the north."

The count() method returns the number of times a specified value appears in the string.

syntax:

string.count(value, start, end)

Value wants to search , start where to search default 0, end string .

Python builtin Data Structures:

1. **List**

list of comma-separated values (items) between square brackets. Lists might contain items of different types, but usually the items all have the same type.

lists can be indexed and sliced.

Lists also support operations like concatenation.

List items are ordered, changeable, and allow duplicate values.

List items are indexed

squares = [1, 4, 9, 16, 25]

**Like strings (and all other built-in sequence types), lists can be indexed and sliced:**

squares[0] # indexing returns the item

1

squares[-1]

25

squares[-3:] # slicing returns a new list

[9, 16, 25]

Lists also support operations like concatenation:

squares + [36, 49, 64, 81, 100]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

**Unlike strings, which are immutable, lists are a mutable type, i.e. it is possible to change their content:**

Example:

cubes = [1, 8, 27, 65, 125] # something's wrong here

4 \*\* 3 # the cube of 4 is 64, not 65!

64

cubes[3] = 64 # replace the wrong value cubes

[1, 8, 27, 64, 125]

You can also add new items at the end of the list, by using the list.append()

cubes.append(216)

**Range()**

list(range(5, 10))

[5, 6, 7, 8, 9]

list(range(0, 10, 3))

[0, 3, 6, 9]

list(range(-10, -100, -30))

[-10, -40, -70]

**a = ['Mary', 'had', 'a', 'little', 'lamb']**

**for i in range(len(a)):**

**print(i, a[i])**

**0 Mary**

**1 had**

**2 a**

**3 little**

**4 lamb**

**Insert Items**

To insert a list item at a specified index, use the insert() method.

The insert() method inserts an item at the specified index:

thislist = ["apple", "banana", "cherry"]

**thislist.insert(1, "orange")**

**thislist.remove("banana")**

**thislist.pop(1)**

**del thislist[0]**

**del thislist**

**thislist.clear()**

**List Comprehension**

**List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.**

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

squares = [x\*\*2 for x in range(10)]

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# filter the list to exclude negative numbers

[x for x in vec if x >= 0]

[0, 2, 4]

# apply a function to all the elements

[abs(x) for x in vec]

[4, 2, 0, 2, 4]

# create a list of 2-tuples like (number, square)

[(x, x\*\*2) for x in range(6)]

[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]

Homogeneous: A homogeneous list or tuple contains elements that are all of the same data type.

A heterogeneous list or tuple contains elements that are of different data types. For example, a heterogeneous list might contain a mix of integers, strings, and booleans.

**Tuples:**

Tuples are immutable.A tuple is a collection which is ordered and unchangeable.and allows duplicate values.

Paking vs Unpaking

unpacking refers to operations involving the assignment of multiple values to a single variable (packing) or extracting multiple values from a data structure (unpacking).

**# Packing into a tuple**

**my\_tuple = 1, 2, 3, 4, 5**

**# Unpacking a tuple**

**a, b, c, d, e = my\_tuple**

**Sets**

Sets are mutable

Python also includes a data type for sets. A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the set() function can be used to create sets. Note: to create an empty set you have to use set(), not {}; the latter creates an empty dictionary.

**basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}**

**print(basket)**

**# show that duplicates have been removed**

**{'orange', 'banana', 'pear', 'apple'}**

**'orange' in basket # fast membership testing**

**True**

**'crabgrass' in basket**

**False**

**# Demonstrate set operations on unique letters from two words**

**a = set('abracadabra')**

**b = set('alacazam')**

**a # unique letters in a**

**{'a', 'r', 'b', 'c', 'd'}**

**a - b # letters in a but not in b**

**{'r', 'd', 'b'}**

**a | b # letters in a or b or both**

**{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}**

**a & b # letters in both a and b**

**{'a', 'c'}**

**a ^ b # letters in a or b but not both**

**{'r', 'd', 'b', 'm', 'z', 'l'}**

**Similarly to list comprehensions, set comprehensions are also supported:**

**a = {x for x in 'abracadabra' if x not in 'abc'}**

**a**

**{'r', 'd'}**

Add Items in Sets:

thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

thisset = {"apple", "banana", "cherry"}

tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

**To remove an item in a set, use the remove(), or the discard() method.**

**If the item to remove does not exist, remove() will raise an error.**

**If the item to remove does not exist, discard() will NOT raise an error.**

**Sets are unordered, so when using the pop() method, you do not know which item that gets removed.**

**dictionaries are indexed by keys, which can be any immutable type.**

**dictionary a set of key: value pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}.**

**The main operations on a dictionary are storing a value with some key and extracting the value given the key.**

dictionaries are mutable data types. This means that you can modify the contents of a dictionary after it has been created. You can add new key-value pairs, update existing key-value pairs, or remove key-value pairs from a dictionary.

Changeable

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

Duplicates Not Allowed

Dictionaries cannot have two items with the same key:

**#both are same get item from dict model is key**

x = thisdict.get("model")

x = thisdict["model"]

**The keys() method will return a list of all the keys in the dictionary.**

thisdict.update({"year": 2020})

thisdict.pop("model")

**The popitem() method removes the last inserted item**

thisdict.popitem()

del thisdict["model"]

**Exercise:**

**List items are ordered, changeable, and allow duplicate values.**

#List Crud

l1 = []#create list

#add items in List

l1.append("a")

l1.insert(1,"b")

l1.insert(1,"c")

l1.insert(2,"c")

x=l1.count("c")

y=l1.index("c")

#update item:

l1[1] = "z"

l1[2:4]=["t","f"]

#delete Items in List

l1.pop() #remove last item from list

l1.remove("a")

del l1[0]

del l1

#Tuple: A tuple is a collection which is ordered and unchangeable allows Duplicate values

#Create Tuple

t1 = ()

# print(type(t1))

#tuple are immutable, and you cannot modify them after creation.

#The append() method is specifically for mutable sequences like lists, not tuples.

# If you need to create a tuple with a single element, you can use a comma after the element enclosed within parentheses:

t2=('a',)

t3=(1,2,3,4,5)

print(t3[2])

# Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.

# But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

x = ("apple", "banana", "cherry")

y = list(x)

y[1] = "kiwi"

x = tuple(y)

**Set**

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

Once a set is created, you cannot change its items, but you can remove items and add new items.

#create set

#list do not allow duplicate values

s1 = {1,2,3,4,5,1,2,3}

s1.add(45) #add item

#delete from list

s1.remove(4)

s1.discard(1)

**#dictionary**

#create dictionary

d={}

#add items in dictionary

d["name"] = "Azhar"

d.update({"Age":23})

#update item in dict

d["name"]="Kashif"

d.update({"Age":45})

#delete items from dict

d.pop("name")

#The popitem() method removes the last inserted item

d.popitem()

#del thisdict["model"]

#del thisdict

print(d)

Difference between modules and classes in Python.

A module in python is simply a way to organize the code, and it contains either python classes or just functions. If you need those classes or functions in your project, you just import them. For instance, the math module in python contains just a bunch of functions, and you just call those needed (math.sin).

Modules are collections of methods and constants. They cannot generate instances. Classes may generate instances (objects), and have per-instance state (instance variables).

Python Lambda Function Syntax

**Syntax: lambda arguments : expression**

This function can have any number of arguments but only one expression, which is evaluated and returned.

One is free to use lambda functions wherever function objects are required.

You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.

It has various uses in particular fields of programming, besides other types of expressions in functions.

Example:

str1 = 'GeeksforGeeks'

upper = lambda string: string.upper()

print(upper(str1))