Immutable Data Structure

Immutable data types are those that can never change their value in place.Immutable is the when no change is possible over time. In Python, if the value of an object cannot be changed over time, then it is known as immutable. Once created, the value of these objects is permanent. Example: String, Tuple etc.

In [1]:
```
#Program showing the example of immutable data structure (string)
greeting = "Welcome to EyeHunts"
greeting[0] = 'Hello'
print(greeting)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [1], line 2
      1 greeting = "Welcome to EyeHunts"
----> 2 greeting[0] = 'Hello'
      3 print(greeting)

TypeError: 'str' object does not support item assignment
```

4.1 Strings

The first immutable data structure is string.

1. String

So What is String? Any sequence of characters within either single quotes or double quotes is considered as a String. Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello". A string is a sequence of characters that can be a combination of letters, numbers, and special characters.

These quotes are not a part of a string, they define only starting and ending of the string. Strings are immutable, i.e., they cannot be changed. Each element of the string can be accessed using indexing or slicing operations.

Syntax for single strings to be printed:

1. a = "Hello"
2. a = 'Hello'

Syntax for assigning multiline string to a variable by using three double quotes or three single quotes:

1. b= """This is the class of immutable data structure

Examples of immutable data structure is String, Tuple, etc."""

2. b= '''This is the class of immutable data structure

Examples of immutable data structure is String, Tuple, etc.'''

In [2]:
```
#Example of printing single line string with single quote:
a='Hello'
print(a)
```

```
Hello
```

In [3]:
```
#Example of printing single line string with double quotes:
a="Hello"
print(a)
```

```
Hello
```

In [5]:
```
#Example of printing multi line string with three single quotes:
b= '''This is the class of immutable data structure
Examples of immutable data structure is String, Tuple, etc.'''
print(b)
```

```
This is the class of immutable data structure
Examples of immutable data structure is String, Tuple, etc.
```

In [6]:
```
#Example of printing multi line string with three double quotes:
b= """This is the class of immutable data structure
Examples of immutable data structure is String, Tuple, etc."""
print(b)
```

```
This is the class of immutable data structure
Examples of immutable data structure is String, Tuple, etc.
```

Accessing characters of string

We can access characters of a string by using the following ways:

1. By using index
2. By using slice operator
3. By using for loop

1. By using index:

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

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

Square brackets can be used to access elements of the string. Python supports both +ve and -ve index.

+ve index means Left to Right (Forward Direction) -ve index means Right to Left (Backward Direction)

In [7]:
```
#Example of accessing the character at position 1 (remember that the first character h

a = "Hello, World!"
print(a[1])
```

```
e
```

In [8]:
```
#Example of accessing the character at position 1 (remember that the first character h
#Backward Direction
a = "Hello, World!"
print(a[-1])
```

```
!
```

In [9]:
```
#Example showing the string index out of range
a = "Hello, World!"
print(a[14])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In [9], line 3
      1 #Example showing the string index out of range
      2 a = "Hello, World!"
----> 3 print(a[14])

IndexError: string index out of range
```

In [10]:
```
#Example showing typeerror in string if float or other string is not allowed.
a = "Hello, World!"
print(a[2.0])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [10], line 3
      1 #Example showing typeerror in string if float or other string is not allowed.
      2 a = "Hello, World!"
----> 3 print(a[2.0])

TypeError: string indices must be integers
```

2. By using slice operator

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.

Get the characters from position 2 to position 5 (not included):

Syntax:

b[beginindex:endindex:step] Begin Index: From where we have to consider slice.

End Index: We have to terminate the slice at endindex-1

Step: Incremented Value.

If we are not specifying begin index then it will consider from beginning of the string.

If we are not specifying end index then it will consider up to end of the string.

The default value for step is 1.

Step value can be either +ve or –ve.

If +ve then it should be forward direction(left to right) and we have to consider begin to end-1.

If -ve then it should be backward direction (right to left) and we have to consider begin to end+1.

print(b[2:5])

In [11]:
```
#Example for slicing for accessing the characters from position 2 to position 5 (not i

b = "Hello, World!"
print(b[2:5])
```

```
llo
```

Slice From the Start

By leaving out the start index, the range will start at the first character:

In [12]:
```
#Example for slicing for accessing the characters from start to position 5 (not inclua

b = "Hello, World!"
print(b[:5])
```

```
Hello
```

Slice To the End

By leaving out the end index, the range will go to the end:

In [13]:
```
#Example for slicing for accessing the characters from position 2, and all the way to

b = "Hello, World!"
print(b[2:])
```

```
llo, World!
```

Negative Indexing

Use negative indexes to start the slice from the end of the string:

In [14]:
```
#Example for slicing for accessing the characters From: "o" in "World!" (position -5)

#To, but not included: "d" in "World!" (position -2):
```

```
b = "Hello, World!"
print(b[-5:-2])
```

```
orl
```

In [16]:
```
##Example for slicing for accessing the characters position 2 to position 10 (not incl

b = "Hello, World!"
print(b[2:10:2])
```

```
lo o
```

In [17]:
```
##Example for slicing for accessing all the characters.

b = "Hello, World!"
print(b[::])
```

```
Hello, World!
```

In [18]:
```
##Example for slicing for accessing all the characters.

b = "Hello, World!"
print(b[:])
```

```
Hello, World!
```

Reversing a String

In order to reverse a string completely. The following syntax is used: b[::-1]

In [19]:
```
##Example for reversing a string completely.

b = "Hello, World!"
print(b[::-1])
```

```
!dlroW ,olleH
```

3. By using for loop.

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

In [20]:
```
#Example for accessing the string through for loop.
for x in "India":
    print(x)
```

```
I
n
d
i
a
```

Concatenation of Strings

To concatenate, or combine, two strings you can use the + operator.

Syntax: c=a+b

To use + operator for strings, compulsory both arguments should be str type.

In [21]:
```
#Example for Merging variable a with variable b into variable c:

a = "Hello"
b = "World"
c = a + b
print(c)
```

```
HelloWorld
```

In [22]:
```
#Example for Merging variable a with variable b into variable c and also adding space

a = "Hello"
b = "World"
c = a +" "+b
print(c)
```

```
Hello World
```

Repition of Strings

To repeat two strings you can use the * operator.

Syntax: c=a*b

To use * operator for strings, compulsory one argument should be str type and other should be int.

In [23]:
```
#Example for Repeating variable a 2 times into variable c:

a = "Hello"

c = a*2
print(c)
```

```
HelloHello
```

Length of a string:

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

Syntax for length of a string:

len(a)

In [24]:
```
#Example for finding the length of a string.
#The len() function returns the length of a string:

a = "Hello, World!"
print(len(a))
```

```
13
```

In [25]:
```
#Example for finding the length of a string.
#The len() function returns the length of a string:

a = "123456789Python"
print(len(a))
```

```
15
```

Capitalize() in String

The capitalize() method returns a string where the first character is upper case, and the rest is lower case.

Syntax

string.capitalize()

In [26]:
```
#Example for capitalizing first letter of the sentence

a = "this is my first class of string"

y = a.capitalize()

print (y)
```

```
This is my first class of string
```

In [27]:
```
#Example for the first character is converted to upper case, and the rest are converte

a = "python is FUN!"
y = a.capitalize()
print (y)
```

```
Python is fun!
```

In [28]:
```
#Example See what happens if the first character is a number:

a="27 is my age"
y=a.capitalize()
print(y)
```

```
27 is my age
```

Enumerate() in String

The enumerate() function is a built-in function that returns an enumerate object. This lets you get the index of an element while iterating Every item in the string is a character. This gives you the character index and the character value, for every character in the string.

If you have a string you can iterate over it with

enumerate(string).

In [29]:
```
#Example for showing enumerate example for a string that prints index along with the c
fruit = "Orange"
for i,j in enumerate(fruit):
    print(i,j)
```

```
0 O
1 r
2 a
3 n
4 g
5 e
```

Enumerate with a Different Starting Index

The enumerate() function has another parameter, the starting index. By default indicides start at zero.

You can change that, lets say you want to start at the second parameter.

enumerate(string,start=index from which one wants to start).

In [30]:
```
#Example for showing enumerate example for a string that prints index along with the c
#required index
fruit = "Orange"
for i,j in enumerate(fruit, start=2):
    print(i,j)
```

```
2 O
3 r
4 a
5 n
6 g
7 e
```

isalnum() in a string

The isalnum() method returns True if all the characters are alphanumeric, meaning alphabet letter (a-z) and numbers (0-9).

Example of characters that are not alphanumeric: (space)!#%&? etc.

Syntax

string.isalnum()

In [32]:
```
#Example for checking whether the string contains alphabet and numeric or not?

a="Apple12"
b="Apple"
c="12345"
d="Apple 12"

print(a.isalnum())
```

```
print(b.isalnum())
print(c.isalnum())
print(d.isalnum())
```

```
True
True
True
False
```

islower() in a string

The islower() method returns True if all the characters are in lower case, otherwise False.

Numbers, symbols and spaces are not checked, only alphabet characters.

Syntax

string.islower()

In [33]:
```
#Example for checking whether the string contains lowercase or uppercase?
a = "Hello world!"
b = "hello 123"
c = "mynameisPeter"

print(a.islower())
print(b.islower())
print(c.islower())
```

```
False
True
False
```

isupper() in string

The isupper() method returns True if all the characters are in upper case, otherwise False.

Numbers, symbols and spaces are not checked, only alphabet characters.

Syntax

string.isupper()

In [34]:
```
#Example Check if all the characters in the texts are in upper case:

a = "Hello World!"
b = "hello 123"
c = "MY NAME IS PETER"

print(a.isupper())
print(b.isupper())
print(c.isupper())
```

```
False
False
True
```

lower() in string

The lower() method returns a string where all characters are lower case.

Symbols and Numbers are ignored.

Syntax

string.lower()

In [35]:
```
#Example of lower case the string:
a = "This is PYTHON"

y = a.lower()

print(y)
```

```
this is python
```

upper() in string

The upper() method returns a string where all characters are in upper case.

Symbols and Numbers are ignored.

Syntax

string.upper()

In [36]:
```
#Example of upper case the string:
a = "This is PYTHON"

y = a.upper()

print(y)
```

```
THIS IS PYTHON
```

isnumeric() in string

The isnumeric() method returns True if all the characters are numeric (0-9), otherwise False.

Exponents, like ² and ¾ are also considered to be numeric values.

"-1" and "1.5" are NOT considered numeric values, because all the characters in the string must be numeric, and the - and the . are not.

Syntax

string.isnumeric()

In [9]:
```
#Example to Check if the characters are numeric:


c = "10km2"
d = "-1"
e = "1.5"
f= "56789"

print(c.isnumeric())
print(d.isnumeric())
print(e.isnumeric())
print(f.isnumeric())
```

```
False
False
False
True
```

find() in string

he find() method finds the first occurrence of the specified value.

The find() method returns -1 if the value is not found.

The find() method is almost the same as the index() method, the only difference is that the index() method raises an exception if the value is not found.

Syntax

string.find(value, start, end)

Parameter Values

Parameter Description

value Required. The value to search for start Optional. Where to start the search. Default is 0 end Optional. Where to end the search. Default is to the end of the string

In [39]:
```
#Example Where in the text is the first occurrence of the letter "e"?:

a = "Hello, Welcome to the Python"

y = a.find("e")

print(y)
```

```
1
```

In [40]:
```
#Example Where in the text is the first occurrence of the letter "e" when you only sea

a = "Hello, Welcome to the Python"
```

```
y = a.find("e",5,10)

print(y)
```

```
8
```

In [42]:
```
#Example If the value is not found, the find() method returns -1, but the index() meth

a = "Hello, Welcome to the Python"

print(a.find("q"))
print(a.index("q"))
```

```
-1
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [42], line 6
      3 a = "Hello, Welcome to the Python"
      5 print(a.find("q"))
----> 6 print(a.index("q"))

ValueError: substring not found
```

rfind() searches the string for a specified value and returns the last position of where it is found.

In [43]:
```
##Example for a rfind() to find the value from last.
txt = "Mi casa, su casa."
x = txt.rfind("casa")
print(x)
```

```
12
```

The index()

The index() method finds the first occurrence of the specified value.

The index() method raises an exception if the value is not found.

The index() method is almost the same as the find() method, the only difference is that the find() method returns -1 if the value is not found.

Syntax

string.index(value, start, end)

Parameter Values

Parameter Description value Required. The value to search for start Optional. Where to start the search. Default is 0 end Optional. Where to end the search. Default is to the end of the string

In [44]:
```
#Example Where in the text is the first occurrence of the letter "e"?:
```

```
a = "Hello, Welcome to the Python"

y = a.index("e")

print(y)
```

```
1
```

In [45]:
```
#Example Where in the text is the first occurrence of the letter "e" when you only sea

a = "Hello, Welcome to the Python"

y = a.index("e",5,10)

print(y)
```

```
8
```

In [46]:
```
#Example If the value is not found, the find() method returns -1, but the index() meth

a = "Hello, Welcome to the Python"

print(a.find("q"))
print(a.index("q"))
```

```
-1
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [46], line 6
      3 a = "Hello, Welcome to the Python"
      5 print(a.find("q"))
----> 6 print(a.index("q"))

ValueError: substring not found
```

Split() in string

The split() method splits a string into a list. The default seperator is space.

You can specify the separator, default separator is any whitespace.

Syntax

string.split(separator)

Parameter Values

Parameter Description separator Optional. Specifies the separator to use when splitting the string. By default any whitespace is a separator

In [47]:
```
#Example of Splitting the string, using comma, followed by a space, as a separator:
txt = "hello, my name is Peter, I am 26 years old"

x = txt.split(", ")
```

```
print(x)
```

```
['hello', 'my name is Peter', 'I am 26 years old']
```

In [48]:
```
#Example of using a hash character as a separator:

txt = "apple#banana#cherry#orange"

x = txt.split("#")

print(x)
```

```
['apple', 'banana', 'cherry', 'orange']
```

In [49]:
```
#Example of splitting if we don't require them in list form finally.

s="Arman Software"
l=s.split()
for x in l:
    print(x)
```

```
Arman
Software
```

strip() in string

The strip() method removes any leading (spaces at the beginning) and trailing (spaces at the end) characters (space is the default leading character to remove)

Syntax

string.strip(characters)

Parameter Values

Parameter Description characters Optional. A set of characters to remove as leading/trailing characters

In [50]:
```
#Example for Removing the leading and trailing characters:
txt = ",,,,,rrttgg.....banana....rrr"

x = txt.strip(",.grt")

print(x)
```

```
banana
```

lstrip() in string

The lstrip() method removes any leading characters (space is the default leading character to remove)

Syntax string.lstrip(characters)

Parameter Values

Parameter Description characters Optional. A set of characters to remove as leading characters

In [51]:
```
#Example for Removing the (leading from left)characters:

txt = ",,,,,ssaaww.....banana"

x = txt.lstrip(",.asw")

print(x)
```

```
banana
```

rstrip() in string

The rstrip() method removes any trailing characters (characters at the end a string), space is the default trailing character to remove.

Syntax

string.rstrip(characters)

Parameter Values

Parameter Description characters Optional. A set of characters to remove as trailing characters

In [52]:
```
#Example for Remove the trailing (leading from right)characters if they are commas, s,

txt = "banana,,,,,ssqqqww....."

x = txt.rstrip(",.qsw")

print(x)
```

```
banana
```

translate() in string

The translate() method returns a string where some specified characters are replaced with the character described in a dictionary, or in a mapping table.

If a character is not specified in the dictionary/table, the character will not be replaced.

If you use a dictionary, you must use ascii codes instead of characters.

Syntax

string.translate(table)

Parameter Values

Parameter Description table Required. Either a dictionary, or a mapping table describing how to perform the replace

In [53]:
```
#Example of Using a mapping table to replace "S" with "P":

txt = "Hello Sam!"
mytable = txt.maketrans("S", "P")
print(txt.translate(mytable))
```

```
Hello Pam!
```

In [54]:
```
#Example of using a mapping table to replace many characters:

txt = "Hi Sam!"
x = "mSa"
y = "eJo"
mytable = txt.maketrans(x, y)
print(txt.translate(mytable))
```

```
Hi Joe!
```

In [55]:
```
#Example of The third parameter in the mapping table describes characters that you wan

txt = "Good night Sam!"
x = "mSa"
y = "eJo"
z = "odnght"
mytable = txt.maketrans(x, y, z)
print(txt.translate(mytable))
```

```
G i Joe!
```

In [56]:
```
#Example of removing punctuation in string using trans() and maketrans()


txt = "Good night,Sam!"

mytable = txt.maketrans(","," ")
print(txt.translate(mytable))
```

```
Good night Sam!
```

isalpha() The isalpha() method returns True if all the characters are alphabet letters (a-z).

Example of characters that are not alphabet letters: (space)!#%&? etc.

Syntax string.isalpha()

In [10]:
```
#Example of isalpha()
txt = "Company10"

x = txt.isalpha()

print(x)
```

```
False
```

In [11]:
```
#Another Example of isalpha()
txt = "CompanyX"

x = txt.isalpha()

print(x)
```

```
True
```

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

Syntax string.count(value, start, end) Parameter Values Parameter Description value Required. A String. The string to value to search for start Optional. An Integer. The position to start the search. Default is 0 end Optional. An Integer. The position to end the search. Default is the end of the string

In [12]:
```
#Example of count() function in string
txt = "I love apples, apple are my favorite fruit"

x = txt.count("apple", 10, 24)

print(x)
```

```
1
```

In [13]:
```
#Another example of count() function in string
txt = "I love apples, apple are my favorite fruit"

x = txt.count("apple")

print(x)
```

```
2
```

The ord() function returns the number representing the unicode code of a specified character.

Syntax ord(character) Parameter Values Parameter Description character String, any character

In [7]:
```
#Example of ord() function
x = ord("h")
print(x)
```

```
104
```

The chr() function returns the character that represents the specified unicode.

Syntax chr(number) Parameter Values Parameter Description number An integer representing a valid Unicode code point

In [8]:
```
#Example of chr() function
x = chr(97)
print(x)
```

```
a
```

4.2 Tuples:

Second Immutable data type is Tuple Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and unchangeable.

Tuples are written with round brackets.

If our data is fixed and never chenges then we should go for Tuple.

Insertion order is preserved.

Duplicates are allowed.

Index plays an important role in Tuple.

Tuple supports both +ve and -ve index means forward direction (from left to right) and -ve index means backward direction (from right to left).

We can represent Tuple elements within paranthesis and with comma seperator.

Paranthesis are optional but recommended to use.

Creation of Tuple:

In [57]:
```
#Example of creation of empty tuple:
t=()
print(type(t))
```

```
<class 'tuple'>
```

Creation of Single Valued Tuple:

A special care has to be taken to compulsory end each value with comma, otherwise it will not be treated as tuple.

In [58]:
```
#Example of creation of single valued tuple:
t=(10,)
p=(10)
print(t)
print(p)
print(type(t))
print(type(p))
```

```
(10,)
10
<class 'tuple'>
<class 'int'>
```

In [59]:
```
#Example of creation of multi value tuple:
t=1,2,3,4
print(t)
print(type(t))
```

```
(1, 2, 3, 4)
<class 'tuple'>
```

In [60]:
```
#Example of creation of Tuple using tuple() Function
t=tuple(range(1,10,2))
print(t)
```

```
(1, 3, 5, 7, 9)
```

Checking Immutability of Tuple

Once we creates tuple, we cannot change its content. Hence tuple objects are immutable. If we try to change a type error will come.

In [61]:
```
#Example of checking immutability
t=1,2,3,4
t[1]=5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [61], line 3
      1 #Example of checking immutability
      2 t=1,2,3,4
----> 3 t[1]=5

TypeError: 'tuple' object does not support item assignment
```

Accessing Elements of Tuple

You can access tuple items by referring to the index number, inside square brackets:

In [62]:
```
#Example of printing the second item in the tuple:

thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

Negative Indexing

Negative indexing means start from the end.

-1 refers to the last item, -2 refers to the second last item etc.

In [63]:
```
#Example of printing the last item of the tuple:

thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

```
cherry
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new tuple with the specified items.

In [64]:
```
#Example and Return the third, fourth, and fifth item:

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

```
('cherry', 'orange', 'kiwi')
```

In [65]:
```
#By leaving out the start value, the range will start at the first item:

#Example
#This example returns the items from the beginning to, but NOT included, "kiwi":

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

```
('apple', 'banana', 'cherry', 'orange')
```

In [66]:
```
#By leaving out the end value, the range will go on to the end of the list:

#Example
#This example returns the items from "cherry" and to the end:

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

```
('cherry', 'orange', 'kiwi', 'melon', 'mango')
```

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

In [67]:
```
#Example this example returns the items from index -4 (included) to index -1 (excludea

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])
```

```
('orange', 'kiwi', 'melon')
```

Deleting a Tuple

As discussed above, we cannot change the elements in a tuple. It means that we cannot delete or remove items from a tuple.

Deleting a tuple entirely, however, is possible using the keyword del.

In [68]:
```
# Example of deleting an item in a tuple
my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# can't delete items

del my_tuple[3]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [68], line 6
      2 my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
      4 # can't delete items
----> 6 del my_tuple[3]

TypeError: 'tuple' object doesn't support item deletion
```

In [69]:
```
# Deleting tuples
my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# Can delete an entire tuple
del my_tuple

# NameError: name 'my_tuple' is not defined
print(my_tuple)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [69], line 8
      5 del my_tuple
      7 # NameError: name 'my_tuple' is not defined
----> 8 print(my_tuple)

NameError: name 'my_tuple' is not defined
```

Concatenation and Repition in Tuple

We can use + operator to combine two tuples. This is called concatenation.

We can also repeat the elements in a tuple for a given number of times using the * operator.

Both + and * operations result in a new tuple.

In [70]:
```
# Example of Concatenation
# Output: (1, 2, 3, 4, 5, 6)
print((1, 2, 3) + (4, 5, 6))

# Example of Repeat
# Output: ('Repeat', 'Repeat', 'Repeat')
print(("Repeat",) * 3)
```

```
(1, 2, 3, 4, 5, 6)
('Repeat', 'Repeat', 'Repeat')
```

Comparing Tuples using >,<,==

Tuples are compared position by position: the first item of the first tuple is compared to the first item of the second tuple; if they are not equal, this is the result of the comparison, else the second item is considered, then the third and so on.

In [71]:
```
#Example of Tuple Comparison
a = (1, 2, 3)
b = (1, 2, 5)
a < b
```

Out[71]:
```
True
```

In [74]:
```
#Second Example of Tuple Comparison
a=(1,2,3)
b=(1,2,3)
a==b
```

Out[74]:
```
True
```

In [75]:
```
#Third Example of Tuple Comparison
a=(1,2,4)
b=(1,2,3)
a>b
```

Out[75]:
```
True
```

In [76]:
```
#Fourth Example of Tuple Comparison
a=(1,2,4)
b=(1,2,3)
a<b
```

Out[76]:
```
False
```

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

Syntax tuple.count(value) Parameter Values Parameter Description value Required. The item to search for

In [6]:
```
#Example of count() function
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.count(5)

print(x)
```

```
2
```

The index() method finds the first occurrence of the specified value.

The index() method raises an exception if the value is not found.

Syntax tuple.index(value) Parameter Values Parameter Description value Required. The item to search for

In [14]:
```
#Example of index()
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.index(8)

print(x)
```

```
3
```

Built in Function- Sorted and reversed

Syntax of sorted() The syntax of the sorted() function is:

sorted(iterable, key=None, reverse=False) sorted() Parameters sorted() can take a maximum of three parameters:

iterable - A sequence (string, tuple, list) or collection (set, dictionary, frozen set) or any other iterator.

reverse (Optional) - If True, the sorted list is reversed (or sorted in descending order). Defaults to False if not provided.

key (Optional) - A function that serves as a key for the sort comparison. Defaults to None.

sorted() Return Value The sorted() function returns a sorted list.

In [77]:
```
#Example of sorting string and tuple
# string
py_string = 'Python'
print(sorted(py_string))

# vowels tuple
py_tuple = ('e', 'a', 'u', 'o', 'i')
print(sorted(py_tuple))
```

```
['P', 'h', 'n', 'o', 't', 'y']
['a', 'e', 'i', 'o', 'u']
```

In [78]:
```
#example of sorting string and tuple in reverse order i.e. descending order
# string
py_string = 'Python'
print(sorted(py_string, reverse=True))

# vowels tuple
py_tuple = ('e', 'a', 'u', 'o', 'i')
print(sorted(py_tuple,reverse=True))
```

```
['y', 't', 'o', 'n', 'h', 'P']
['u', 'o', 'i', 'e', 'a']
```

min() inbuilt function The min() function returns the item with the lowest value, or the item with the lowest value in an iterable.

If the values are strings, an alphabetically comparison is done.

Syntax min(n1, n2, n3, ...) Or: min(iterable) Parameter Values Parameter Description n1, n2, n3, ... One or more items to compare Or: Parameter Description iterable An iterable, with one or more items to compare

In [2]:
```
#example of min function
x = min("Mike", "John", "Vicky")
print(x)
```

```
John
```

In [3]:
```
#another example of min function
x=min(1454,267,454)
print(x)
```

```
267
```

The max() function returns the item with the highest value, or the item with the highest value in an iterable.

If the values are strings, an alphabetically comparison is done.

Syntax max(n1, n2, n3, ...) Or: max(iterable) Parameter Values Parameter Description n1, n2, n3, ... One or more items to compare Or: Parameter Description iterable An iterable, with one or more items to compare

In [4]:
```
#example of max function
x = max("Mike", "John", "Vicky")
print(x)
```

```
Vicky
```

In [5]:
```
#another example of max function
x=max(1454,267,454)
print(x)
```

```
1454
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: