Tuples

- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is ordered and unchangeable and allow duplicate values.
- Tuples are written with round brackets.
- tuple are read only data type but it has not write functionality

- **Ordered :** When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.
- **Unchangeable :** Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.
- **Allow Duplicates :** Since tuples are indexed, they can have items with the same value:

- create
- access
- edit
- add
- delete
- operations
- functions

1. Create

In [1]:

```
# how to create a tuple
T1=()
```

In [2]:

```
T2=(1,2,3,4)
T2
# homogenous tuple
```

Out[2]:

```
(1, 2, 3, 4)
```

In [3]:

```
T3=("hello",4,5.6,True)
T3
# heterogenous tuple
```

Out[3]:

```
('hello', 4, 5.6, True)
```

In [4]:

```
T4=(1,2,3,(4,5))
# 2D tuple
T4
```

Out[4]:

```
(1, 2, 3, (4, 5))
```

- **Note :**

  - we can't make single object tuple, it will show in as a class of variable

In [5]:

```
T5=(1)
print(T5)
type(T5)
```

```
1
```

Out[5]:

```
int
```

In [6]:

```
T5=("himanshu")
type(T5)
# srtill it will not become as a tuple
```

Out[6]:

```
str
```

- **Create Tuple With One Item**

  - To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

In [7]:

```
T5=("hello",)
T5
```

Out[7]:

```
('hello',)
```

- **The tuple() Constructor**

    - It is also possible to use the tuple() constructor to make a tuple.

In [8]:

```
thistuple = tuple(("apple", "banana", "cherry"))
# note the double round-brackets
print(thistuple)
```

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

In [9]:

```
T6=tuple("goa")
T6
```

Out[9]:

```
('g', 'o', 'a')
```

In [10]:

```
T6=tuple([1,2,3,4])
T6
# it will make a tuple of the items inside the lists
```

Out[10]:

```
(1, 2, 3, 4)
```

In [11]:

```
# Tuples allow duplicate values:
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

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

2. Access

- You can access tuple items by referring to the index number, inside square brackets:
- **Note :** The first item has index 0.

In [12]:

```
T2=(1,2,3,4)
T2
```

Out[12]:

```
(1, 2, 3, 4)
```

In [13]:

```
T2[0]
# for indexing
```

Out[13]:

```
1
```

In [14]:

```
''' By leaving out the start value,
the range will start at the first item:'''
T2[:4]
```

Out[14]:

```
(1, 2, 3, 4)
```

In [16]:

```
'''By leaving out the end value,
the range will go on to the end of the list:'''

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

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

In [17]:

```
T4=(1,2,3,(4,5))
T4
```

Out[17]:

```
(1, 2, 3, (4, 5))
```

In [18]:

```
T4[-1][1]
```

Out[18]:

```
5
```

- **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 [20]:

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

'''Note: The search will start at index 2 (included) and
end at index 5 (not included).'''
```

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

Out[20]:

```
'Note: The search will start at index 2 (included) and \nend at index 5 (n
ot included).'
```

In [22]:

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

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

'''This example returns the items from index -4 (included)
to index -1 (excluded)'''
```

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

Out[22]:

```
'This example returns the items from index -4 (included) \nto index -1 (ex
cluded)'
```

3. Edit

- Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created.

In [23]:

```
L=[1,2,3,4,5]
L[0]=100
L
# editing is easy in the list
```

Out[23]:

```
[100, 2, 3, 4, 5]
```

In [24]:

```
T2=(1, 2, 3, 4)
T2
```

Out[24]:

```
(1, 2, 3, 4)
```

In [25]:

```
T2[0]=100
# it will throw an error because tuple does not support the editing
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_10744\3555532746.py in <module>
----> 1 T2[0]=100
      2 # it will throw an error because tuple does not support the editin
g

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

- Change Tuple Values

> - 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.

In [26]:

```
# Convert the tuple into a list to be able to change it:

x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

```
('apple', 'kiwi', 'cherry')
```

4. Add

- Since tuples are immutable, they do not have a build-in append() method.
- tuples just like strings are immutable
- we can not add new item in the existing tuple

In [28]:

```
'''Convert the tuple into a list, add "orange",
and convert it back into a tuple:'''

thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
print(thistuple)
```

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

5. Delete

- The del keyword can delete the tuple completely:

In [29]:

```
T1=()
T1
```

Out[29]:

```
()
```

In [30]:

```
del T1
```

In [31]:

```
T1
# this will throw and error because T1 is already deleted
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_10744\889919135.py in <module>
----> 1 T1
      2 # this will throw and error because T1 is already deleted

NameError: name 'T1' is not defined
```

In [32]:

```
T2=(1, 2, 3, 4)
T2
```

Out[32]:

```
(1, 2, 3, 4)
```

In [33]:

```
del T2[-1]
# we can not delete a single item from the tuple because it is immutable
# also we can not delete a part of tuple
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_10744\1119365915.py in <module>
----> 1 del T2[-1]
      2 # we can not delete a single item from the tuple because it is imm
utable
      3 # also we can not delete a part of tuple

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

In [35]:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple)
#this will raise an error because the tuple no longer exists
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_10744\3513908944.py in <module>
      1 thistuple = ("apple", "banana", "cherry")
      2 del thistuple
----> 3 print(thistuple)
      4 #this will raise an error because the tuple no longer exists

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

6. Operations

In [36]:

```
T2=(1, 2, 3, 4)
T2
```

Out[36]:

```
(1, 2, 3, 4)
```

In [37]:

```
T3=('hello', 4, 5.6, True)
T3
```

Out[37]:

```
('hello', 4, 5.6, True)
```

- **Join Two Tuples**

  - To join two or more tuples you can use the + operator:

In [38]:

```
T2+T3
# this is concatination
```

Out[38]:

```
(1, 2, 3, 4, 'hello', 4, 5.6, True)
```

In [39]:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

In [40]:

```
# Create a new tuple with the value "orange", and add that tuple:

thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

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

- **Multiply Tuples**

  - If you want to multiply the content of a tuple a given number of times, you can use the * operator:

In [41]:

```
print(T2*3)
# this is multiplication
```

```
(1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4)
```

In [42]:

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

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

- **Loop Through a Tuple**

  - You can loop through the tuple items by using a for loop.

In [43]:

```
# we can run a loop
T2=(1,2,3,4,5,6,7,8,9,10)
for i in T2:
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

- **Loop Through the Index Numbers**

  - You can also loop through the tuple items by referring to their index number.
  - Use the range() and len() functions to create a suitable iterable.

In [44]:

```
# Print all items by referring to their index number:

thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
    print(thistuple[i])
```

```
apple
banana
cherry
```

- **Using a While Loop**

- You can loop through the tuple items by using a while loop.
- Use the len() function to determine the length of the tuple, then start at 0 and loop your way through the tuple items by referring to their indexes.
- Remember to increase the index by 1 after each iteration.

In [45]:

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
    print(thistuple[i])
    i = i + 1
```

```
apple
banana
cherry
```

- **membership operators in tuple**

In [46]:

```
T2=(1, 2, 3, 4)
1 in T2
```

Out[46]:

```
True
```

In [47]:

```
T3=('hello', 4, 5.6, True)
5 in T3
```

Out[47]:

```
False
```

In [48]:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
    print("Yes, 'apple' is in the fruits tuple")
```

```
Yes, 'apple' is in the fruits tuple
```

- **Unpacking a Tuple**

- When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

In [49]:

```
# Packing a tuple:
fruits = ("apple", "banana", "cherry")
fruits
```

Out[49]:

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

> - But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

Note :

> - The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.

In [50]:

```
# Unpacking a tuple:

fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)
```

```
apple
banana
cherry
```

- **Using Asterisk ***

> - If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list:

In [51]:

```
# Assign the rest of the values as a list called "red":

fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

```
apple
banana
['cherry', 'strawberry', 'raspberry']
```

- If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

In [52]:

```
# Add a list of values the "tropic" variable:

fruits = ("apple", "mango", "papaya", "pineapple", "cherry")
(green, *tropic, red) = fruits

print(green)
print(tropic)
print(red)
```

```
apple
['mango', 'papaya', 'pineapple']
cherry
```

7. Functions

- To determine how many items a tuple has, use the len() function:

In [53]:

```
T2=(1, 2, 3, 4)
T2
```

Out[53]:

```
(1, 2, 3, 4)
```

In [54]:

```
len(T2)
```

Out[54]:

```
4
```

In [55]:

```
min(T2)
```

Out[55]:

```
1
```

In [56]:

```
max(T2)
```

Out[56]:

```
4
```

In [57]:

```
sum(T2)
```

Out[57]:

```
10
```

In [58]:

```
sorted(T2,reverse=True)\
# it will be converted in the list
```

Out[58]:

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

In [48]:

```
# tuple is immutable
# list is a mutable
```