Tuples

- Tuples are used to hold together multiple objects. Think of them as similar to lists, but without the extensive functionality that the list class gives you. One major feature of tuples is that they are immutable like strings i.e. you cannot modify tuples.
- Tuples are defined by specifying items separated by commas within an optional pair of parentheses.
- Tuples are usually used in cases where a statement or a user-defined function can safely assume that the collection of values i.e. the tuple of values used will **not change**.

```
# The intern at NASA

planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn",
"Uranus", "Neptune"]

type(planets)

list

id(planets)

140456308451264

planets[2] = "Rahul"

planets

['Mercury', 'Venus', 'Rahul', 'Mars', 'Jupiter', 'Saturn', 'Uranus',
'Neptune']

id(planets)

140456308451264


# Immutable

planets = ("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn",
"Uranus", "Neptune")

type(planets)

tuple

id(planets)

140455095373376

# Tuples supports indexing
```

```
planets[2]
```

```
'Earth'
```

```
planets[2] = "Rahul"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/zn/hkv6562d6_d30glfs8yc76900000gn/T/ipykernel_16733/726823221.py in <module>
----> 1 planets[2] = "Rahul"

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

```
# In tuples you cant change the values
```

```
# Hence tuples are immutable
```

```
# Making tuple
```

```
# 1st way
```

```
t = ("Rahul")
```

```
type(t)
```

```
str
```

```
(2 * 3) + (5 * 6)
```

```
36
```

```
t = ("Rahul",)
```

```
type(t)
```

```
tuple
```

```
# 2nd way
```

```
t1 = ()
```

```
type(t1)
```

```
tuple
```

```
len(t1)
```

```
0
```

```
# 3rd way
# Quiz
t2 = tuple("Rahul")
t2
```

```
('R', 'a', 'h', 'u', 'l')
```

```
type(t2)
```

```
tuple
```

```
# Quiz
tuple('hello world')
```

```
('h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd')
```

```
# tuple(234)
# print(dir(234))
t3 = (2,)
t3
```

```
(2,)
```

```
len(t3)
```

```
1
```

```
# Iteration, indexing and slicing
# Ordered --> Indexing
t = tuple("Rahul")
```

```
t
('R', 'a', 'h', 'u', 'l')
t[0]
'R'
t[1]
'a'
t[2]
'h'
# Tuple supports slicing as well
t[::]
('R', 'a', 'h', 'u', 'l')
exaple = t[::-1]
exaple
('l', 'u', 'h', 'a', 'R')
t
('R', 'a', 'h', 'u', 'l')
t = t[::-1]
t
('R', 'a', 'h', 'u', 'l')

# Count and index
t.count("A")
0
t.count("a")
1
t.index("a")
1
li = [12, 13, 14, 15]
```

```
li1 = li[::-1]
li
[12, 13, 14, 15]
li1
[15, 14, 13, 12]

print(type((1,2,3)))
<class 'tuple'>

t = tuple("Rahul")
t
('R', 'a', 'h', 'u', 'l')
id(t)
140455099222352
t1 = t[::-1]
t1
('l', 'u', 'h', 'a', 'R')
id(t1)
140455099222272
t = t[::-1]
id(t)
140455240187392
t
('l', 'u', 'h', 'a', 'R')
```

Sets

Problem: Suppose that you are working as a data analyst at an edtech company. The edtech company is offering courses on Calculus(C) and Linear Algebra(L) among many others. Now you want to represent the students in the courses. Which data structure to use?

- Assume all names are unique for the students taking part in the edtech company. We have done this for better understanding. We could have taken id which will be unique.

Options?

- Lists => It can contain duplicate values
- Tuples => It can also contain duplicate values.

There is a different alternative => Sets

```
students = ["Reetu", "Shahzer", "Rahul", "Reetu"]
```

```
students
```

```
['Reetu', 'Shahzer', 'Rahul', 'Reetu']
```

```
t = tuple(students)
```

```
t
```

```
('Reetu', 'Shahzer', 'Rahul', 'Reetu')
```

```
s = {"Reetu", "Shahzer", "Rahul", "Reetu", "Rahul"}
```

```
s
```

```
{'Rahul', 'Reetu', 'Shahzer'}
```

```
len(s)
```

```
3
```

```
# Sets are unordered data structure
```

```
# in operator
```

```
s
```

```
{'Rahul', 'Reetu', 'Shahzer'}
```

```
"Rahul" in s
```

```
True
```

```
# Set does not support indexing
```

Creation

- set()
- set(iterable)

```
s = {}
```

```
type(s)
```

```
dict
```

```
# We store data using {} in 2 DS
# Dict and sets
```

```
s = set()
```

```
type(s)
```

```
set
```

```
s
```

```
set()
```

```
s = set("Rahul janghu")
```

```
s
```

```
{' ', 'R', 'a', 'g', 'h', 'j', 'l', 'n', 'u'}
```

```
# Quiz
```

```
set('hello world')
```

```
{' ', 'd', 'e', 'h', 'l', 'o', 'r', 'w'}
```

Iteration and indexing?

- We cannot access a set item by referring to an index or a key. If you cannot access an element we cannot modify it. That's why it is also unchangeable
- We can however run a loop to iterate.

```
# s[0]

s

{' ', 'R', 'a', 'g', 'h', 'j', 'l', 'n', 'u'}

# Sets does not support indexing

# print(dir(s))

for i in s:
    print(i)

u
a
R
g
l
h

n
j
```

```
# Challenge: Count number of unique words in a sentence

sent = "be the change you wish to see in the world"

# set(sent)

li = sent.split()

li

['be', 'the', 'change', 'you', 'wish', 'to', 'see', 'in', 'the',
'world']

len(li)

10

s = set(li)

s

{'be', 'change', 'in', 'see', 'the', 'to', 'wish', 'world', 'you'}
```

```
len(s)
```

```
9
```

Changing a set

- add: For single element
- update(iterable)

```
# add

# Quiz

packing = {"Laptop", "Camera", "Clothes", "charger", "water"}

packing

{'Camera', 'Clothes', 'Laptop', 'charger', 'water'}

len(packing)

5

type(packing)

set

packing.add("Snacks")

packing.add("books")

packing.add("mobile")

packing

{'Camera',
 'Clothes',
 'Laptop',
 'Snacks',
 'books',
 'charger',
 'mobile',
 'water'}



# update

toilet_kit = ["Soap", "Shampoo", "Toothbrush", "Toothpaste", "Deo"]

toilet_kit
```

```
['Soap', 'Shampoo', 'Toothbrush', 'Toothpaste', 'Deo']
```

```
packing
```

```
{'Camera',
 'Clothes',
 'Laptop',
 'Snacks',
 'books',
 'charger',
 'mobile',
 'water'}
```

```
packing.update(toilet_kit)
```

```
print(packing)
```

```
{'charger', 'mobile', 'books', 'Clothes', 'Snacks', 'Laptop', 'Deo',
'Soap', 'Toothpaste', 'water', 'Shampoo', 'Toothbrush', 'Camera'}
```

```
s
```

```
{'be', 'change', 'in', 'see', 'the', 'to', 'wish', 'world', 'you'}
```

```
# Sets are mutable type of DataStructure
```

Deleting an element

- pop: removes random element. We are not sure what it is
- remove(element): Removes particular element

```
# pop:
# It deletes data randomly
# return the data
```

```
packing
```

```
{'Camera',
 'Clothes',
 'Deo',
 'Laptop',
 'Shampoo',
 'Snacks',
 'Soap',
 'Toothbrush',
 'Toothpaste',
 'books',
 'charger',
```

```
 'mobile',
 'water'}

deleted = packing.pop()

deleted

'mobile'

packing

{'Camera',
 'Clothes',
 'Deo',
 'Laptop',
 'Shampoo',
 'Snacks',
 'Soap',
 'Toothbrush',
 'Toothpaste',
 'books',
 'water'}



# remove
# It accepts a value that you want to remove
# It does not return the deleted element

packing

{'Camera',
 'Clothes',
 'Deo',
 'Laptop',
 'Shampoo',
 'Snacks',
 'Soap',
 'Toothbrush',
 'Toothpaste',
 'books',
 'water'}

deleted = packing.remove("Camera")

deleted

packing

{'Clothes',
 'Deo',
 'Laptop',
 'Shampoo',
```

```
 'Snacks',
 'Soap',
 'Toothbrush',
 'Toothpaste',
 'books',
 'water'}
```

```
packing.remove("Snacks")
```

```
packing
```

```
{'Clothes',
 'Deo',
 'Laptop',
 'Shampoo',
 'Soap',
 'Toothbrush',
 'Toothpaste',
 'books',
 'water'}
```

```
# Quizzes
```

```
l = [1,1,2,2,3,3]
s = set(l)
print(len(s))
```

```
3
```

```
# 1, 2, 3
```

```
# s = set()
# s.append(5)
```

Intersection

- Suppose you want to find out which students are enrolled in both the Calculus and Linear Algebra Course. Then you can use the intersection method.

```
# Common in both sets
```

```
calculus = {"Shahzer", "Ritvik", "Mutthu", "Rahul"}
```

```
calculus
```

```
{'Mutthu', 'Rahul', 'Ritvik', 'Shahzer'}
```

```
linear = {"Anjali", "Ritvik", "Aniket", "Rahul"}
linear
{'Aniket', 'Anjali', 'Rahul', 'Ritvik'}
calculus.intersection(linear)
{'Rahul', 'Ritvik'}
linear.intersection(calculus)
{'Rahul', 'Ritvik'}
```

Union

- Suppose you want to find out which students are enrolled in either the Calculus or the Linear Algebra Course or in both. Then you can use the union method.

```
# All elements in both sets
linear
{'Aniket', 'Anjali', 'Rahul', 'Ritvik'}
calculus
{'Mutthu', 'Rahul', 'Ritvik', 'Shahzer'}
linear.union(calculus)
{'Aniket', 'Anjali', 'Mutthu', 'Rahul', 'Ritvik', 'Shahzer'}
calculus.union(linear)
{'Aniket', 'Anjali', 'Mutthu', 'Rahul', 'Ritvik', 'Shahzer'}
```

Difference

- Suppose you want to find out the set of students who have enrolled in the Calculus course but not in Linear Algebra course or vice-versa, then we can use the difference method.

```
linear
{'Aniket', 'Anjali', 'Rahul', 'Ritvik'}
calculus
```

```
{'Mutthu', 'Rahul', 'Ritvik', 'Shahzer'}

linear - calculus

{'Aniket', 'Anjali'}

calculus - linear

{'Mutthu', 'Shahzer'}
```

```
# Quizzes

a = {1,2,3}
b = {3,4,5}
print(a-b)
print(a.union(b))
print(a.intersection(b))

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

# 1 2
# 1, 2, 3, 4, 5
# 3

set1 = {1, 2, 3, 4, 5, 6}
set2 = {2, 4, 5, 6, 7}
x = set1 - set2

x

{1, 3}
```

```
# Doubts
```