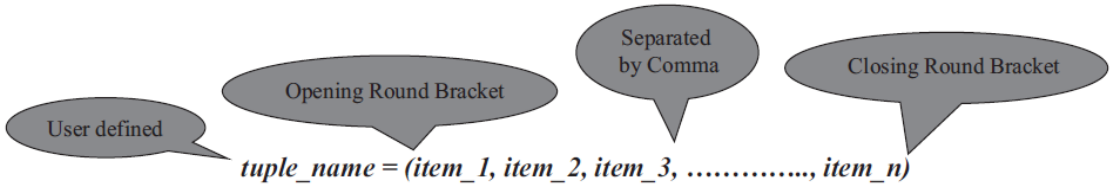
Creating the Tuples

The syntax for creating the tuple is

```
tuple_name = (item_1, item_2, item_3, ..... , item_n)
```



In [ ]:
```
#Creating Tuples
internet = ("cern", "timbernerslee", "www", 1980)
internet
```

Out[ ]:
```
('cern', 'timbernerslee', 'www', 1980)
```

In [ ]:
```
type(internet)
```

Out[ ]:
```
tuple
```

In [ ]:
```
f1 = "ferrari", "redbull", "mercedes", "williams", "renault"
f1
```

Out[ ]:
```
('ferrari', 'redbull', 'mercedes', 'williams', 'renault')
```

In [ ]:
```
type(f1)
```

Out[ ]:
```
tuple
```

In [ ]:
```
# creating empty tuple
empty_tuple = ()
empty_tuple
```

Out[ ]:
```
()
```

In [ ]:
```
#Can store any item of type string, number, object,another_variable, tuple
air_force = ("f15", "f22a", "f35a")
fighter_jets = (1988, 2005, 2016, air_force)
fighter_jets
```

Out[ ]:
```
(1988, 2005, 2016, ('f15', 'f22a', 'f35a'))
```

In [ ]:
```
# A tuple with one item is constructed by having a value followed by a comma
singleton = 'hello',
singleton
```

Out[ ]:
```
('hello',)
```

In [ ]:
```
#Basic Tuple Operations
tuple_1 = (2, 0, 1, 4)
tuple_2 = (2, 0, 1, 9)
tuple_1 + tuple_2
```

Out[ ]:
```
(2, 0, 1, 4, 2, 0, 1, 9)
```

In [ ]:
```
tuple_1 * 3
```

Out[ ]:
```
(2, 0, 1, 4, 2, 0, 1, 4, 2, 0, 1, 4)
```

In [ ]:
```
print(tuple_1 == tuple_2)
print(tuple_1 > tuple_2)
```

```
False
False
```

In [ ]:
```
tuple_1 != tuple_2
```

Out[ ]:
```
True
```

In [ ]:
```
#The built-in tuple() function is used to create a tuple
norse = "vikings"
string_to_tuple = tuple(norse)
string_to_tuple
```

Out[ ]:
```
('v', 'i', 'k', 'i', 'n', 'g', 's')
```

In [ ]:
```
zeus = ["g", "o", "d", "o", "f", "s", "k", "y"]
list_to_tuple = tuple(zeus)
list_to_tuple
```

Out[ ]:
```
('g', 'o', 'd', 'o', 'f', 's', 'k', 'y')
```

In [ ]:
```
string_to_tuple + tuple("scandinavia")
```

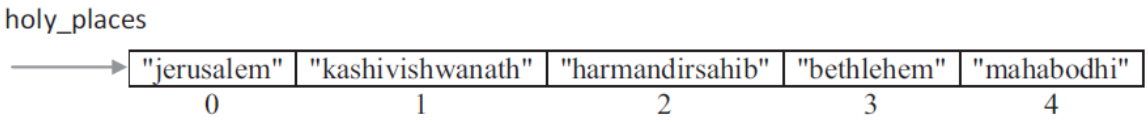
Out[ ]:
```
('v',
 'i',
 'k',
 'i',
 'n',
 'g',
 's',
 's',
 'c',
 'a',
 'n',
 'd',
 'i',
 'n',
 'a',
 'v',
 'i',
 'a')
```

In [ ]:
```
list_to_tuple + tuple(["g", "r", "e", "e", "k"])
```

Out[ ]:
```
('g', 'o', 'd', 'o', 'f', 's', 'k', 'y', 'g', 'r', 'e', 'e', 'k')
```

In [ ]:
```
#Nexted Tuples
letters = ("a", "b", "c")
numbers = (1, 2, 3)
nested_tuples = (letters, numbers)
print(nested_tuples)
tuple("wolverine")
```

Out[ ]:
```
(('a', 'b', 'c'), (1, 2, 3))
('w', 'o', 'l', 'v', 'e', 'r', 'i', 'n', 'e')
```



In [ ]:
```
#Indexing and Slicing in Tuples
holy_places = ("jerusalem", "kashivishwanath", "harmandirsahib", "bethlehem","mahabo
holy_places
```

Out[ ]:
```
('jerusalem', 'kashivishwanath', 'harmandirsahib', 'bethlehem', 'mahabodhi')
```

In [ ]:
```
print(holy_places[0])
```

```
jerusalem
```

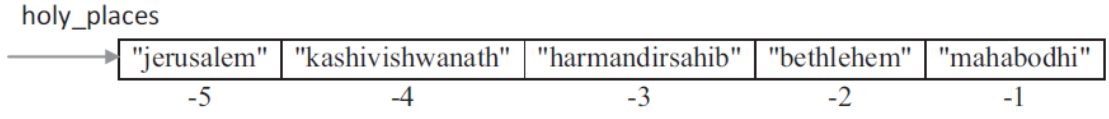
In [ ]:
```
print(holy_places[1])
print(holy_places[2])
print(holy_places[3])
print(holy_places[4])
```

```
kashivishwanath
harmandirsahib
bethlehem
mahabodhi
```

In [ ]:
```
holy_places[1] = "Mathura"
```

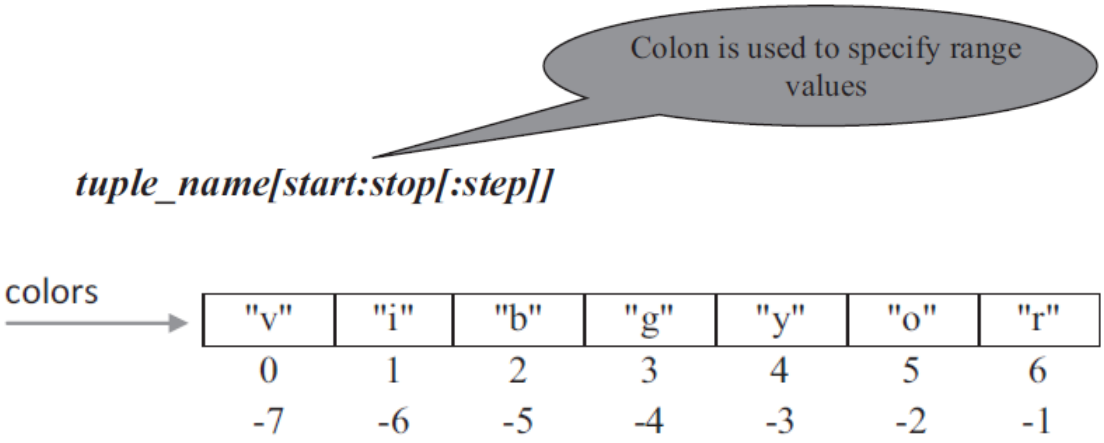
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-9879619a27cc> in <module>
----> 1 holy_places[1] = "Mathura"

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



In [ ]:
```
#Negative Indexing in Tuples
holy_places[-2]
```

Out[ ]: 'bethlehem'



In [ ]:
```
#Slicing in Tuples
colors = ("v", "i", "b", "g", "y", "o", "r")
colors
```

Out[ ]: ('v', 'i', 'b', 'g', 'y', 'o', 'r')

In [ ]:
```
colors[1:4]
```

Out[ ]: ('i', 'b', 'g')

In [ ]:
```
colors[:5]
```

Out[ ]: ('v', 'i', 'b', 'g', 'y')

In [ ]:
```
colors[3:]
```

Out[ ]: ('g', 'y', 'o', 'r')

In [ ]:
```
colors[:]
```

Out[ ]: ('v', 'i', 'b', 'g', 'y', 'o', 'r')

In [ ]:
```
colors[::]
```

Out[ ]: ('v', 'i', 'b', 'g', 'y', 'o', 'r')

In [ ]:
```
colors[1:5:2]
```

Out[ ]: ('i', 'g')

In [ ]:
```
colors[::2]
```

Out[ ]: ('v', 'b', 'y', 'r')

In [ ]:

```
colors[::-1]
```

Out[ ]:
```
('r', 'o', 'y', 'g', 'b', 'i', 'v')
```

In [ ]:
```
colors[-5:-2]
```

Out[ ]:
```
('b', 'g', 'y')
```

Built-In Functions Used on Tuples

Built-In Functions	Description
len()	The *len()* function returns the numbers of items in a tuple.
sum()	The *sum()* function returns the sum of numbers in the tuple.
sorted()	The *sorted()* function returns a sorted copy of the tuple as a list while leaving the original tuple untouched.

In [ ]:
```
#Built-In Functions on Tuples
years = (1987, 1985, 1981, 1996)
print(len(years))
print(sum(years))
sorted_years = sorted(years)
print(sorted_years)
```

```
4
7949
[1981, 1985, 1987, 1996]
```

In [ ]:
```
#Relationship between Tuples and Lists
coral_reef = ("great_barrier", "ningaloo_coast", "amazon_reef", "pickles_reef")
coral_reef_list = list(coral_reef)
coral_reef_list
```

Out[ ]:
```
['great_barrier', 'ningaloo_coast', 'amazon_reef', 'pickles_reef']
```

In [ ]:
```
#An item within a tuple is mutable
german_cars = ["porsche", "audi", "bmw"]
european_cars = ("ferrari", "volvo", "renault", german_cars)
european_cars
```

Out[ ]:
```
('ferrari', 'volvo', 'renault', ['porsche', 'audi', 'bmw'])
```

In [ ]:
```
european_cars[3].append("mercedes")
print(german_cars)
print(european_cars)
```

```
['porsche', 'audi', 'bmw', 'mercedes', 'mercedes', 'mercedes']
('ferrari', 'volvo', 'renault', ['porsche', 'audi', 'bmw', 'mercedes', 'mercedes',
'mercedes'])
```

In [ ]:
```
#Relation between Tuples and Dictionaries
fish_weight_kg = (("white_shark", 520), ("beluga", 1571), ("greenland_shark",1400))
fish_weight_kg_dict = dict(fish_weight_kg)
fish_weight_kg_dict
```

Out[ ]: {'white_shark': 520, 'beluga': 1571, 'greenland_shark': 1400}

Various Tuple Methods

Tuple Methods	Syntax	Description
count()	tuple_name.count(item)	The *count()* method counts the number of times the item has occurred in the tuple and returns it.
index()	tuple_name.index(item)	The *index()* method searches for the given item from the start of the tuple and returns its index. If the value appears more than once, you will get the index of the first one. If the item is not present in the tuple, then *ValueError* is thrown by this method.

Note: Replace the word *"tuple_name"* mentioned in the syntax with your *actual tuple name* in your code.

In [ ]:
```
channels = ("ngc", "discovery", "animal_planet", "history", "ngc")
print(channels.count("ngc"))
print(channels.index("history"))
```

```
2
3
```

In [ ]:
```
#Tuple Packing and UnPacking
t= 12345, 54321, 'hello!'
t
```

Out[ ]: (12345, 54321, 'hello!')

In [ ]:
```
x, y, z = t
print(x)
print(y)
print(z)
```

```
12345
54321
hello!
```

In [ ]:
```
# Program 8.1: Program to Iterate Over Items in Tuples Using for Loop

ocean_animals = ("electric_eel", "jelly_fish", "shrimp", "turtles", "blue_whale")


def main():
    for each_animal in ocean_animals:
        print(f"{each_animal} is a ocean animal")


if __name__ == "__main__":
    main()
```

```
electric_eel is a ocean animal
jelly_fish is a ocean animal
shrimp is a ocean animal
turtles is a ocean animal
blue_whale is a ocean animal
```

In [ ]:
```
# Program 8.2: Program to Populate Tuple with User-Entered Items

tuple_items = ()

total_items = int(input("Enter the total number of items: "))
```

```
for i in range(total_items):
    user_input = int(input("Enter a number: "))
    tuple_items += (user_input,)
print(f"Items added to tuple are {tuple_items}")


list_items = []
total_items = int(input("Enter the total number of items: "))
for i in range(total_items):
    item = input("Enter an item to add: ")
    list_items.append(item)

items_of_tuple = tuple(list_items)

print(f"Tuple items are {items_of_tuple}")
```

```
Enter the total number of items: 4
Enter a number: 4
Enter a number: 7
Enter a number: 8
Enter a number: 3
Items added to tuple are (4, 7, 8, 3)
Enter the total number of items: 3
Enter an item to add: 5
Enter an item to add: 79
Enter an item to add: 5
Tuple items are ('5', '79', '5')
```

In [ ]:

```
# Program 8.3: Write Python Program to Swap Two Numbers Without Using
# Intermediate/Temporary Variables. Prompt the User for Input


def main():
    a = int(input("Enter a value for first number "))
    b = int(input("Enter a value for second number "))

    b, a = a, b

    print("After Swapping")
    print(f"Value for first number {a}")
    print(f"Value for second number {b}")


if __name__ == "__main__":
    main()
```

```
Enter a value for first number 45
Enter a value for second number 23
After Swapping
Value for first number 23
Value for second number 45
```

In [ ]:

```
# Program 8.4: Program to Demonstrate the Return of Multiple Values from a Function


def return_multiple_items():
    monument = input("Which is your favorite monument? ")
    year = input("When was it constructed? ")
    return monument, year


def main():
    mnt, yr = return_multiple_items()
```

```
    print(f"My favorite monument is {mnt} and it was constructed in {yr}")

    result = return_multiple_items()
    print(f"My favorite monument is {result[0]} and it was constructed in {result[1]

if __name__ == "__main__":
    main()
```

```
Which is your favorite monument? taj
When was it constructed? agra
My favorite monument is taj and it was constructed in agra
Which is your favorite monument? rr
When was it constructed? 456
My favorite monument is rr and it was constructed in 456
```

In [ ]:

```
# Program 8.5: Write Python Program to Sort Words in a Sentence in Decreasing
# Order of Their Length. Display the Sorted Words along with Their Length


def sort_words(user_input):
    list_of_words = user_input.split()
    words = list()
    for each_word in list_of_words:
        words.append((len(each_word), each_word))

    words.sort(reverse=True)

    print("After sorting")
    for length, word in words:
        print(f'The word "{word}" is of {length} characters')


def main():
    sentence = input("Enter a sentence ")
    sort_words(sentence)


if __name__ == "__main__":
    main()
```

```
Enter a sentence this is veena
After sorting
The word "veena" is of 5 characters
The word "this" is of 4 characters
The word "is" is of 2 characters
```

In [ ]:

```
# Program 8.6: Write Pythonic Code to Sort a Sequence of Names according
# to Their Alphabetical Order Without Using sort() Function


def read_list_items():
    print("Enter names separated by a space")
    list_items = input().split()
    return list_items


def sort_item_list(items_in_list):
    n = len(items_in_list)
    for i in range(n):
        for j in range(1, n-i):
            if items_in_list[j-1] > items_in_list[j]:
                (items_in_list[j-1], items_in_list[j]) = (items_in_list[j], items_in
```

```
    print("After Sorting")
    print(items_in_list)


def main():
    all_items = read_list_items()
    sort_item_list(all_items)


if __name__ == "__main__":
    main()
```

```
Enter names separated by a space
veena vijay
After Sorting
['veena', 'vijay']
```

In [ ]:

```
#Using zip function
x = [1, 2, 3]
y = [4, 5, 6]
zipped = zip(x, y)
list(zipped)
```

Out[ ]:

```
[(1, 4), (2, 5), (3, 6)]
```

In [ ]:

```
#To loop over two or more sequences at the same time, the entries can be paired with
questions = ('name', 'quest', 'favorite color')
answers = ('lancelot', 'the holy grail', 'blue')
for q, a in zip(questions, answers):
    print(f'What is your {q}? It is {a}')
```

```
What is your name? It is lancelot
What is your quest? It is the holy grail
What is your favorite color? It is blue
```

In [ ]:

```
# A set is an unordered collection with no duplicate items.
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket)
print('orange' in basket)
print('crabgrass' in basket)
```

```
{'apple', 'pear', 'orange', 'banana'}
True
False
```

In [ ]:

```
a = set('abracadabra')
b = set('alacazam')
print(a)
print(b)
```

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

In [ ]:

```
print(a-b)
print(a|b)
print(a&b)
print(a^b)
print(len(basket))
print(sorted(basket))
```

```
{'d', 'r', 'b'}
{'r', 'm', 'b', 'c', 'd', 'a', 'l', 'z'}
{'a', 'c'}
{'r', 'd', 'b', 'm', 'l', 'z'}
4
['apple', 'banana', 'orange', 'pear']
```

Various Set Methods

Set Methods	Syntax	Description
add()	set_name.add(*item*)	The *add()* method adds an *item* to the set *set_name*.
clear()	set_name.clear()	The *clear()* method removes all the items from the set *set_name*.
difference()	set_name.difference(**others*)	The *difference()* method returns a new set with items in the set *set_name* that are not in the *others* sets.
discard()	set_name.discard(item)	The *discard()* method removes an *item* from the set *set_name* if it is present.
intersection()	set_name.intersection(**others*)	The *intersection()* method returns a new set with items common to the set *set_name* and all *others* sets.
isdisjoint()	set_name.isdisjoint(*other*)	The *isdisjoint()* method returns True if the set *set_name* has no items in common with *other* set. Sets are disjoint if and only if their intersection is the empty set.
issubset()	set_name.issubset(*other*)	The *issubset()* method returns True if every item in the set *set_name* is in *other* set.
issuperset()	set_name.issuperset(*other*)	The *issuperset()* method returns True if every element in *other* set is in the set *set_name*.
pop()	set_name.pop()	The method *pop()* removes and returns an arbitrary item from the set set_name. It raises KeyError if the set is empty.
remove()	set_name.remove(*item*)	The method *remove()* removes an *item* from the set *set_name*. It raises KeyError if the *item* is not contained in the set.
symmetric_ difference()	set_name. symmetric_difference(other)	The method *symmetric_difference()* returns a new set with items in either the set or other but not both.
union()	set_name.union(**others*)	The method *union()* returns a new set with items from the set *set_name* and all *others* sets.
update()	set_name.update(**others*)	Update the set *set_name* by adding items from all *others* sets.

Note: Replace the words "*set_name*", "*other*" *and* "*others*" mentioned in the syntax with your *actual set names* in your code.

In [ ]:
```
#Set Methods examples
european_flowers = {"sunflowers", "roses", "lavender", "tulips", "goldcrest"}
american_flowers = {"roses", "tulips", "lilies", "daisies"}
american_flowers.add("orchids")
print(american_flowers.difference(european_flowers))
print(american_flowers.intersection(european_flowers))
print(american_flowers.isdisjoint(european_flowers))
print(american_flowers.issuperset(european_flowers))
```

```
{'orchids', 'daisies', 'lilies'}
{'roses', 'tulips'}
False
False
```

In [ ]:
```
print(american_flowers.issubset(european_flowers))
print(american_flowers.symmetric_difference(european_flowers))
print(american_flowers.union(european_flowers))
american_flowers.update(european_flowers)
print(american_flowers)
```

```
False
{'orchids', 'daisies', 'sunflowers', 'lavender', 'goldcrest', 'lilies'}
{'roses', 'daisies', 'lilies', 'tulips', 'orchids', 'lavender', 'sunflowers', 'goldc
rest'}
```

```
{'roses', 'daisies', 'lilies', 'tulips', 'orchids', 'lavender', 'sunflowers', 'goldc
rest'}
```

In [ ]:

```
american_flowers.discard("roses")
print(american_flowers)
print(european_flowers.pop())
american_flowers.clear()
print(american_flowers)
```

```
{'daisies', 'lilies', 'tulips', 'orchids', 'lavender', 'sunflowers', 'goldcrest'}
roses
set()
```

In [ ]:

```
# Program 8.7: Program to Iterate Over Items in Sets Using for Loop

warships = {"u.s.s._arizona", "hms_beagle", "ins_airavat", "ins_hetz"}


def main():
    for each_ship in warships:
        print(f"{each_ship} is a Warship")


if __name__ == "__main__":
    main()
```

```
ins_hetz is a Warship
hms_beagle is a Warship
u.s.s._arizona is a Warship
ins_airavat is a Warship
```

In [ ]:

```
# Program 8.8: Write a Function Which Receives a Variable Number of
# Strings as Arguments. Find Unique Characters in Each String


def find_unique(*all_words):
    for each_word in all_words:
        unique_character_list = list(set(each_word))
        print(f"Unique characters in the word {each_word} are {unique_character_list


def main():
    find_unique("egg", "immune", "feed", "vacuum", "goddessship")


if __name__ == "__main__":
    main()
```

```
Unique characters in the word egg are ['g', 'e']
Unique characters in the word immune are ['u', 'i', 'n', 'e', 'm']
Unique characters in the word feed are ['f', 'e', 'd']
Unique characters in the word vacuum are ['a', 'u', 'v', 'c', 'm']
Unique characters in the word goddessship are ['g', 'i', 's', 'p', 'e', 'd', 'h',
'o']
```

In [ ]:

```
# Program 8.9: Write a Python Program That Accepts a Sentence as Input
# and Removes All Duplicate Words. Print the Sorted Words


def unique_words(user_input):
    words = user_input.split()
```

```
    print(f"The unique and sorted words are {sorted(list(set(words)))}")


def main():
    sentence = input("Enter a sentence ")
    unique_words(sentence)


if __name__ == "__main__":
    main()
```

```
Enter a sentence This is a beautiful city
The unique and sorted words are ['This', 'a', 'beautiful', 'city', 'is']
```

In [ ]:
```
#frozenset
#A frozenset is basically the same as a set, except that it is immutable. Once a fro
fs = frozenset(["g", "o", "o", "d"])
fs
```

Out[ ]:
```
frozenset({'d', 'g', 'o'})
```

In [ ]:
```
animals = set([fs, "cattle", "horse"])
animals
```

Out[ ]:
```
{'cattle', frozenset({'d', 'g', 'o'}), 'horse'}
```

In [ ]:
```
official_languages_world = {"english":59, "french":29, "spanish":21}
frozenset(official_languages_world)
```

Out[ ]:
```
frozenset({'english', 'french', 'spanish'})
```

In [ ]:
```
frs = frozenset(["german"])
official_languages_world = {"english":59, "french":29, "spanish":21, frs:6}
official_languages_world
```

Out[ ]:
```
{'english': 59, 'french': 29, 'spanish': 21, frozenset({'german'}): 6}
```

Tuple Exercise Write a program to read email IDs of n number of students and store them in a tuple. Create two new tuples, one to store only the usernames from the email IDs and second to store domain names from the email ids. Print all three tuples at the end of the program. [Hint: You may use the function split()]

In [ ]:
```
emails = tuple()
username = tuple()
domainname = tuple()
#Create empty tuple 'emails', 'username' and domain-name
n = int(input("How many email ids you want to enter?: "))
for i in range(0,n):
    emid = input("> ")
    #It will assign emailid entered by user to tuple 'emails'
    emails = emails +(emid,)
    #This will split the email id into two parts, username and domain and return a l
    spl = emid.split("@")
    #assigning returned list first part to username and second part to domain name
    username = username + (spl[0],)
    domainname = domainname + (spl[1],)
```

```
print("\nThe email ids in the tuple are:")
#Printing the list with the email ids
print(emails)

print("\nThe username in the email ids are:")
#Printing the list with the usernames only
print(username)

print("\nThe domain name in the email ids are:")
#Printing the list with the domain names only
print(domainname)
```

```
The email ids in the tuple are:
('veenamehtry@gmail.com', 'veenamehtry@yahoo.co.in')

The username in the email ids are:
('veenamehtry', 'veenamehtry')

The domain name in the email ids are:
('gmail.com', 'yahoo.co.in')
```

Tuple Exercise Buffet A buffet-style restaurant offers only five basic foods. Think of five simple foods, and store them in a tuple. 1.Use a for loop to print each food the restaurant offers. 2.Try to modify one of the items, and make sure that Python rejects the change. 3.The restaurant changes its menu, replacing two of the items with different foods. Add a block of code that rewrites the tuple, and then use a for loop to print each of the items on the revised menu.

In [ ]:

```
Buffet = ("paneer starters","platter","main course", "Veg Noodles", "Veg Manchurian"
for each_fooditem in Buffet:
        print(f"{each_fooditem} is a Buffet-style food item")
Buffet[0] = "Masala Pappad"
print(Buffet)
```

```
paneer starters is a Buffet-style food item
platter is a Buffet-style food item
main course is a Buffet-style food item
Veg Noodles is a Buffet-style food item
Veg Manchurian is a Buffet-style food item
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-c9f281b5a0a1> in <module>
      2 for each_fooditem in Buffet:
      3         print(f"{each_fooditem} is a Buffet-style food item")
----> 4 Buffet[0] = "Masala Pappad"
      5 print(Buffet)

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

In [ ]:

Tuple Exercises Blocks of Stock.

A block of stock as a number of attributes, including a purchase date, a purchase price, a number of shares, and a ticker symbol. We can record these pieces of information in a tuple for each block of stock and do a number of simple operations on the blocks. Let's dream that we have the following portfolio.

Purchase Date	Purchase Price	Shares	Symbol	:Current Price”
25 Jan 2001	43.50	25	CAT	92.45
25 Jan 2001	42.80	50	DD	51.19
25 Jan 2001	42.10	75	EK	34.87
25 Jan 2001	37.58	100	GM	37.58

Hint: Represent it as a tuple.

We can represent each block of stock as a 5-tuple with purchase date, purchase price, shares, ticker symbol and current price.

portfolio= [("25-Jan-2001", 43.50, 25, 'CAT', 92.45), ("25-Jan-2001", 42.80, 50, 'DD', 51.19), ("25-Jan-2001", 42.10, 75, 'EK', 34.87), ("25-Jan-2001", 37.58, 100, 'GM', 37.58)]

1. Develop a function that examines each block, multiplies shares by purchase price and determines the total purchase price of the portfolio.
2. Develop a second function that examines each block, multiplies shares by purchase price and shares by current price to determine the total amount gained or lost.

In [ ]:
```
portfolio= [ ( "25-Jan-2001", 43.50, 25, 'CAT', 92.45 ),
( "25-Jan-2001", 42.80, 50, 'DD', 51.19 ),
( "25-Jan-2001", 42.10, 75, 'EK', 34.87 ),
( "25-Jan-2001", 37.58, 100, 'GM', 37.58 )
]
print(portfolio[0])
print(portfolio[1])
print(portfolio[2])
print(portfolio[3])
```

```
('25-Jan-2001', 43.5, 25, 'CAT', 92.45)
('25-Jan-2001', 42.8, 50, 'DD', 51.19)
('25-Jan-2001', 42.1, 75, 'EK', 34.87)
('25-Jan-2001', 37.58, 100, 'GM', 37.58)
```

In [ ]: