# Tuples in Python


Tuples are another cool data type in Python.


They are very similar to `lists`…except that you **cannot change what is contained inside a** tuple *once it has been created*.

Rules for **tuples**:

1. You **cannot** add elements to a tuple. Tuples have no append or extend methods.
2. You **cannot** remove elements from a tuple. Tuples have no remove or pop method.
3. You *can* find elements in a tuple, since this doesn't change the tuple.
4. You *can* also use the `in` operator to check if an element exists in the tuple.

Now is a good time to discuss some commonly used termonology in computer science (regardless of programming language).


**Mutable** - Can be changed or modified. Like a list or a set.

**Immutable** - Can **NOT** be changed or modified (like a tuple).

**Iterable** - A data type *capable of returning its members one at a time* (strings, lists, sets, and tuples are all iterable).

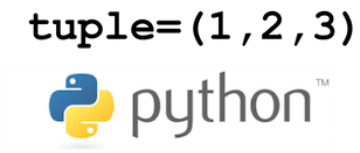
**Non-iterable** – (integer-int, character-char, decimal number-floats, are all non-iterable data types).

## So why use tuples?

Right now we are at a stage in our programming that may not require tuples…. but tuples are useful and commonly used by experience programmers for the following reasons:

- **Tuples are faster than lists (they use less memory).** If you're defining a constant set of values and that don't change….use a tuple.
- Tuples **make data safer**. Data contained in a tuple is **"write-protected"** data that can't be changed.
- Some tuples can be used as **dictionary** keys. We will learn about dictionaries in the next assignment. Stay tuned!

# When to use tuples?




For now:

**Use tuples** when you have you want to *group bits of data together* that are **directly** related to each other.

**Example#1:**

```
student1=("Jeff",1999)
student2=("Tim",2003)
student3=("Dave",2005)
student4=("Mary",2002)
```

Does it make sense to store a 5th value of someone's birthday **without** their name? NO!...so use a tuple!

`Student5=(2002)?` **this record wouldn't make sense…**

If the relationship between two or more pieces of data doesn't change (a person's name and the year they were born) the data might be seen as one (immutable) element and tuples might be the way to go.

**Example#2:** `name, employee#, job description`

```
tuple1 = ("John", 23567, "Software Engineer")
tuple2 = ('Kevin', '23568', 'Manager')
tuple2 = ('Sandra', '23569', 'CEO')
tuple2 = ('Mary', '23590', 'Software Developer')
```

**Example#3:**

Notice tuples use round parenthesis ( )

```
thistuple = ("apple", "banana", "cherry")
```

and we can use an index to access info:

```
print(thistuple[0])
```

"apple"

# Exercise#1

To access values in tuple, use the square brackets for ***slicing*** along with the ***index*** to obtain value available at that index. For example:

**Try out the following code**:

```
tuple1=(0 ,1, 2, 3)

print(tuple1[1:])
print(tuple1[::-1])
print(tuple1[2:4])

tuple2 = ('python', 'geek')
print(len(tuple2))
```

# Exercise#2

```
Tuple1 = (2, 4, 3, 5, 4, 6, 7, 8, 6, 1)
```

Use the tuple above, and write 4 separate mini python programs to:

a) Find the length of a tuple above.
b) Print the 3rd element in the tuple.
c) Find out how many 6's are in the tuple.
d) Find out the max value of the tuple.

# Exercise#3

With tuples, we can also extract and manipulation data quite easily (identical to the way we can with lists).

**Examine** the code below.
**Predict** what it will do.
**Try it** out in Trinket (or another IDE).

```
student1=("Jeff",1999)
student2=("Tim",2003)
student3=("Dave",2005)
student4=("Mary",2002)

class1=[student1,student2,student3,student4]

print(class1)
print(student1[1])
print(class1[3])
```

Using the same tuples and list above, try the code below and make sure you know what is happening and why:

```
new_total=0
for x in class1:
  new_total=new_total+x[1]
print((new_total)/len(class1))
```

Answer: *the above code should give use the* ***average year*** *of births of our class1*

We can also get a class list without birth years:

Try the following:

```
for x in class1:
  print(x[0])
```

# Exercise#4

We can also **"unpack"** data in a tuple in an organized way by assigning each **index** in the tuple a **variable name**:

**Examine**,
**Predict**,
and **Try** out the following code:

```
#weather data recorded on different days:

w1=("raining",16,"partly cloudy",22)
w2=("sunny",22,"no clouds",10)
w3=("sunny",21,"partly cloudy",9)
w4=("raining",17,"cloudy",4)
w5=("raining",11,"cloudy",2)
```

**# print the weather data for day 2**

```
print(w2)
```

**# assign each index in a tuple a variable name.**

```
(perciptation,temp,cloudcover,wind)=w4
```

**# print the precipitation data for w4 (day 4) .**

```
print(precipitation)
```

**#put all the weather data into a list.**

```
weatherdata=(w1,w2,w3,w4,w5)
```

**#print out all the temperature data for the recorded days.**

```
for x in weatherdata:
  (precipitation,temp,cloudcover,wind)=x
  print(temp)
```

**Your turn!**

Find the average temperature for the weather day's recorded.

# Exercise#5:

You have a list of (x,y,z) coordinates they represent points in a three dimensional space. In fact, they represent different positons in this classroom! Unfortunately, a giant steps on the classroom and squishes it so that each coordinates now all have a height of **zero** making `z=0` in each tuple.

### Part A

Write a Python program to replace *last* value of each **tuple** in a **list**.
You will have to convert the tuples into lists first
Make sure you turn them back into tuples one you've changed the z-coordinates

```
Sample list=[(10, 20, 40), (40, 50, 60), (70, 80, 90),(20, 30 ,10)]
```

### Part B

Now,

Find out and print the maximum y value in the list of tuples.
Find out and print the maximum and minimum x values in the list of tuples.

# Exercise#6:

Write a Python program to sort a list of tuples by the size of its **float** element from lowest value to highest value.

Your output should be a list in the correct order

Sample Input:

```
list=[('item1', 1.20), ('item2', 15.10), ('item3', 24.5),
('itme4', 2.67)]
```

Sample Output:

```
list=[('item1', 1.20), ('itme4', 2.67),('item2', 15.10),
('item3', 24.5),]
```

# Exercise#6: Stock Prices

The stocks you own have a number of attributes, including: a purchase date, a purchase price, a number of shares you own, and a ticker symbol. We can record these pieces of information in a **`tuple`** for each stock you own!

Let's dream that we have the following portfolio.

| Purchase Date | Purchase Price | Shares (you owned) | Symbol | Sold 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 |

We can represent our portfolio with a list of **`tuples`**. Each 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 )]
```

Develop a program that examines your portfolio by:

- Multiplying the number shares by **purchase price** of each stock, and determines the total purchase price of the entire portfolio.
- Multiplying the number shares by **sold price** of each stock, and determines the total current value of your portfolio.
- Determines the **total amount** you gained or lost in your **portfolio.**

Develop a second program that:

- Multiplies number of shares of **each** stock by **purchase** price.
- Then multiplies the number of shares of **each** stock by ***sold price.***
- Determines the amount of money gained or lost by ***each*** stock.
- Put the list in order of most profitable to least profitable stock.

# Exercise#7

Create a list of `(x,y)` coordinate **`tuples`**. Then create a program that can add the x and y values form one coordinate with the corresponding x and y values of a second coordinate of the user's choice.

**Sample input:**

```
(1,2)
(3,4)
(9,5)
(0,3)
(2,1)

What coordinates do you want ADD?
2 and 4
```

**Sample output:**

```
(3,4) + (0,3) = (3,7)
```

# Exercise#8

Write a function that takes a list of two sets of data that need to be paired and creates a list of **`tuples`** that pairs the data

**Sample input:**

```
 ([10, 20,30], 'abc'),
```

**Sample output:**

the result will be….. `[(10, 'a'), (20, 'b'), (30, 'c')].`

# Exercise#9

Take data from an **external** `text file` that holds: a group of numbers representing each hour in the day and an average of how may emails are coming into someone's mailbox:

```
1 am       0
2 am      0
3 am       0
4 am       0
5 am       0
6 am       1
7 am       6
8 am       7
9 am       14
10 am      11
12 pm       4
1 pm        3
2 pm        4
3 pm        6
4 pm        8
5 pm        12
6 pm        5
7 pm        2
8 pm        1
9 pm        1
10 pm       1
11 pm       1
12 pm       0
```

Now sort the data in to a list of **`tuples`** (from greatest number of emails to least).

**Example:**

```
Emails= [('9 am', 14), ('5 pm', 12), ('10 pm', 11),....]
```

# Exercise#10

Create a simple text based game that:

> a) Describes what each of the following terms mean in computer science,... and then
> b) Quizzes the user on their meaning and which data types can be described by which terms.

Mutable
Immutable
Iterable
Non-iterable