Introduction 🧑‍💻

Our misson today is to cover:

- What is an array & why we need it
- How might you use an array
- How to write to and read from an array
- Helpful methods that can be used with arrays
- Quiz

I will use python for the code snippets

What is an array?

A general definition:

> " ...array, is a data structure consisting of a **collection of elements** (values or variables), each identified by at least one array index or key... " (from wikipedia's 'array' page) (https://en.wikipedia.org/wiki/Array_data_structure)

Why do we need arrays?

Why is a structured collection of elements needed?

Why don't we just create a reference for every variable?

With or without array - is there a difference?

See below example where we test out both approaches - with and without the use of an array.

We are asked to process some raw data about employees' salary.
Requirement is : `1)print out the salaries of each employee to console` and then `2)calculate the sum of all salaries.`
The data given to us is like this:

```
Employee id, salary
0, 5000
1, 6000
2, 5000
3, 5500
4, 4000
```

Approach 1 - without array 🤷🏻‍♂️

In [1]:

```python
# Without array

employee0 = 5000
employee1 = 6000
employee2 = 5000
employee3 = 5500
employee4 = 4000

print(f'Salary of employee0 is { employee0 }')
print(f'Salary of employee1 is { employee1 }')
print(f'Salary of employee2 is { employee2 }')
print(f'Salary of employee3 is { employee3 }')
print(f'Salary of employee4 is { employee4 }')

salaries_sum = employee0 + employee1 + employee2 + employee3 + employee4
print(f'Sum of all salaries is { salaries_sum }')
```

```
Salary of employee0 is 5000
Salary of employee1 is 6000
Salary of employee2 is 5000
Salary of employee3 is 5500
Salary of employee4 is 4000
Sum of all salaries is 25500
```

> Imagine when there are 5000 employees instead of just 5?

> We can use an array to group the data. Then we are able to process them with higher efficiency.

Approach 2 - with array 🙆‍♂️

In [2]:

```
# With array (list)

salaries = [5000, 6000, 5000, 5500, 4000]

# a loop to print each entry
for i, salary in enumerate(salaries):
    print(f'Salary of Employee{ i } is { salary }')

# make use of python's sum()
print(f'Sum of all salaries is { sum(salaries) }')
```

```
Salary of Employee0 is 5000
Salary of Employee1 is 6000
Salary of Employee2 is 5000
Salary of Employee3 is 5500
Salary of Employee4 is 4000
Sum of all salaries is 25500
```

> Only 4 lines of code to achieve the same result, no matter how many employees join later!

"Array" in Python 🐍

some terminology

In python, the most commonly used array-like data structure is `list` .

In this session, we will focus on learning the basics of python `list` .

(Python does have `array` as a module and it is part of the python standard library.)

How do we use a list in python?

- Basic operations
- List methods
- Built-in functions related to list

Basic Operations - create, read and write

In [3]:

```
# Declaration

salaries = [5000, 6000, 5000, 5500, 4000]

empty_list = []

mixed_types = ['zero', 1, 2]
```

In [4]:

```
# Read element

salaries = [5000, 6000, 5000, 5500, 4000]

print(salaries[0])

print(salaries[-1]) # negative indexing

# print(salaries[5]) # IndexError
```

```
5000
4000
```

In [5]:

```
# Write element

salaries = [5000, 6000, 5000, 5500, 4000]

salaries[0] = 5001

print(f'salary for Employee0 is now { salaries[0] }')
```

```
salary for Employee0 is now 5001
```

List Methods

Python includes following list methods:

list.append(obj) - Appends object obj to list

list.count(obj) - Returns count of how many times obj occurs in list

list.extend(seq) - Appends the contents of seq to list

list.index(obj) - Returns the lowest index in list that obj appears

list.insert(index, obj) - Inserts object obj into list at offset index

list.pop(obj=list[-1]) - Removes and returns last object or obj from list

list.remove(obj) - Removes object obj from list

list.reverse() - Reverses objects of list in place

list.sort([func]) - Sorts objects of list, use compare func if given

In [6]:

```
# list.append()

odd_numbers = [1, 3, 5]

odd_numbers.append(7)

print(odd_numbers)
```

```
[1, 3, 5, 7]
```

In [7]:

```
# list.extend()

odd_numbers = [1, 3, 5]

odd_numbers.extend([7, 9, 11, 13])

print(odd_numbers)
```

```
[1, 3, 5, 7, 9, 11, 13]
```

In [8]:

```
# we can also use + operator to combine two list,
# it is called 'concatenation'

odd_numbers = [1, 3, 5]

print(odd_numbers + [7, 9, 11, 13])
```

```
[1, 3, 5, 7, 9, 11, 13]
```

In [9]:

```
# list.remove()

odd_numbers = [1, 3, 4, 5]

odd_numbers.remove(4)

print(odd_numbers)
```

```
[1, 3, 5]
```

Python also has built-in functions that can be used on `list`

A few functions for example:

> len(list) - Gives the total length of the list
>
> max(list) and min(list) - Returns item from the list with max/min value
>
> sum(list) Sums up the numbers in the list
>
> ......

In [10]:

```
# len(list)

salaries = [5000, 6000, 5000, 5500, 4000]

print(len(salaries))
```

```
5
```

In [11]:

```
# max(list), min(list)

salaries = [5000, 6000, 5000, 5500, 4000]

print(max(salaries))

print(min(salaries))
```

```
6000
4000
```

In [12]:

```
# sum(list)

salaries = [5000, 6000, 5000, 5500, 4000]

print(sum(salaries))
```

```
25500
```

Questions? 🤔

Exercise 🏋️🏋️🏋️

You are asked to calculate the **average** salary of all employees from three companies.

The data are given to you as three arrays `company_one` , `company_two` and `company_three` .

Note that there is a mistake made by the person who collected the data. The last entry in `company_two` was accidentally put as **negative.** Correct this mistake before you calculate the average salary.

company_one = [6000, 6500, 7000, 5000, 6400]
company_two = [5000, 5700, 7300, -5900] (-5900 needs to be changed to 5900)
company_three = [6500, 7000, 4000, 5200, 5700, 5700, 8300]

In [13]:

```
# data
company_one = [6000, 6500, 7000, 5000, 6400]
company_two = [5000, 5700, 7300, -5900]
company_three = [6500, 7000, 4000, 5200, 5700, 5700, 8300]

# your solution here:
```

More To Explore

The actual array module in python:

> https://docs.python.org/3/library/array.html (https://docs.python.org/3/library/array.html)

Some other stuff we can do with `list`:

In [14]:

```
# 2-D list

two_d_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(two_d_list[2])

print(two_d_list[2][2])
```

```
[7, 8, 9]
9
```

In [15]:

```
# list slicing using the slicing operator [:]
# syntax: [S:E] --> getting elements from index S to index E-1

my_list = ['a','b', 'c', 'd', 'e', 'f', 'g']

# elements from index 2 to index 4
print(my_list[2:5])

# elements from index 5 to end
print(my_list[5:])

# elements beginning to end
print(my_list[:])
```

```
['c', 'd', 'e']
['f', 'g']
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```