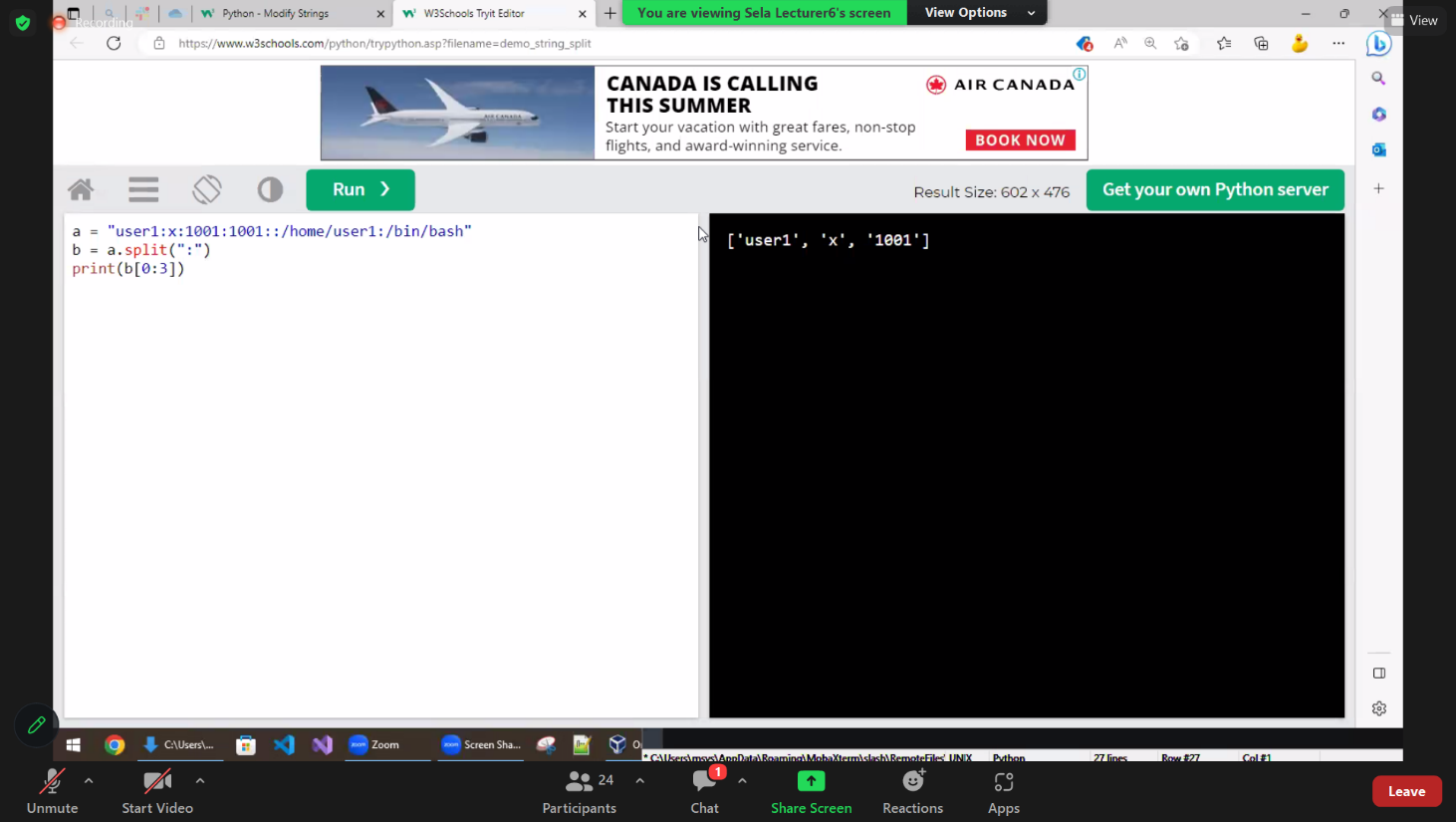
Python summary-



4 built-in **data types** in Python used to **store collections of data**-

[List](https://www.w3schools.com/python/python_lists.asp)-

[Set](https://www.w3schools.com/python/python_sets.asp)-

[Dictionary](https://www.w3schools.com/python/python_dictionaries.asp)-

Tuple-

**Global variables** can be used by everyone, both inside of functions and outside-

global x

Get the first character of the string txt.

txt = "Hello World"

x = 

txt [0]

Get the characters from index 2 to index 4 (llo).

txt = "Hello World"

x = 

txt[2:5]

## List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are [Tuple](https://www.w3schools.com/python/python_tuples.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

Lists are created using square brackets:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

thislist.pop(1)

gets the second in the list (the first argument is slot 0)

thislist.remove(1) / (“banana”)

len(thislist) - length of list

String, int and boolean data types:

list1 = ["apple", "banana", "cherry"]  
list2 = [1, 5, 7, 9, 3]  
list3 = [True, False, False]

From Python's perspective, lists are defined as objects with the data type 'list'

Thislist.pop – removes last slot of list

For I in range(20)

Thislist.append(I)

# it makes a list with int 0-19

Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.

Return the third, fourth, and fifth item:

print(thislist[2:5])

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

This example returns the items from "cherry" to the end:

thislist[2:]

Insert "watermelon" as the third item:

thislist.insert(2, "watermelon")

thislist = ["apple", "banana", "cherry"]

print(thislist[1])

“banana”

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

Only accept items that are not "apple":

newlist = [x for x in fruits if x != "apple"]

Sort the list alphabetically: also numerically  
thislist.sort()

Thislist.copy() – creats a copy

Range(len(mylist)) – goes through all list #0,1,2….

Convert my list to capital

For I in range(len(mylist))

Mylist[i] = mylist[i}.upper()

for I in range(5, 10, 2) - from 5 to 10 in hops of 2

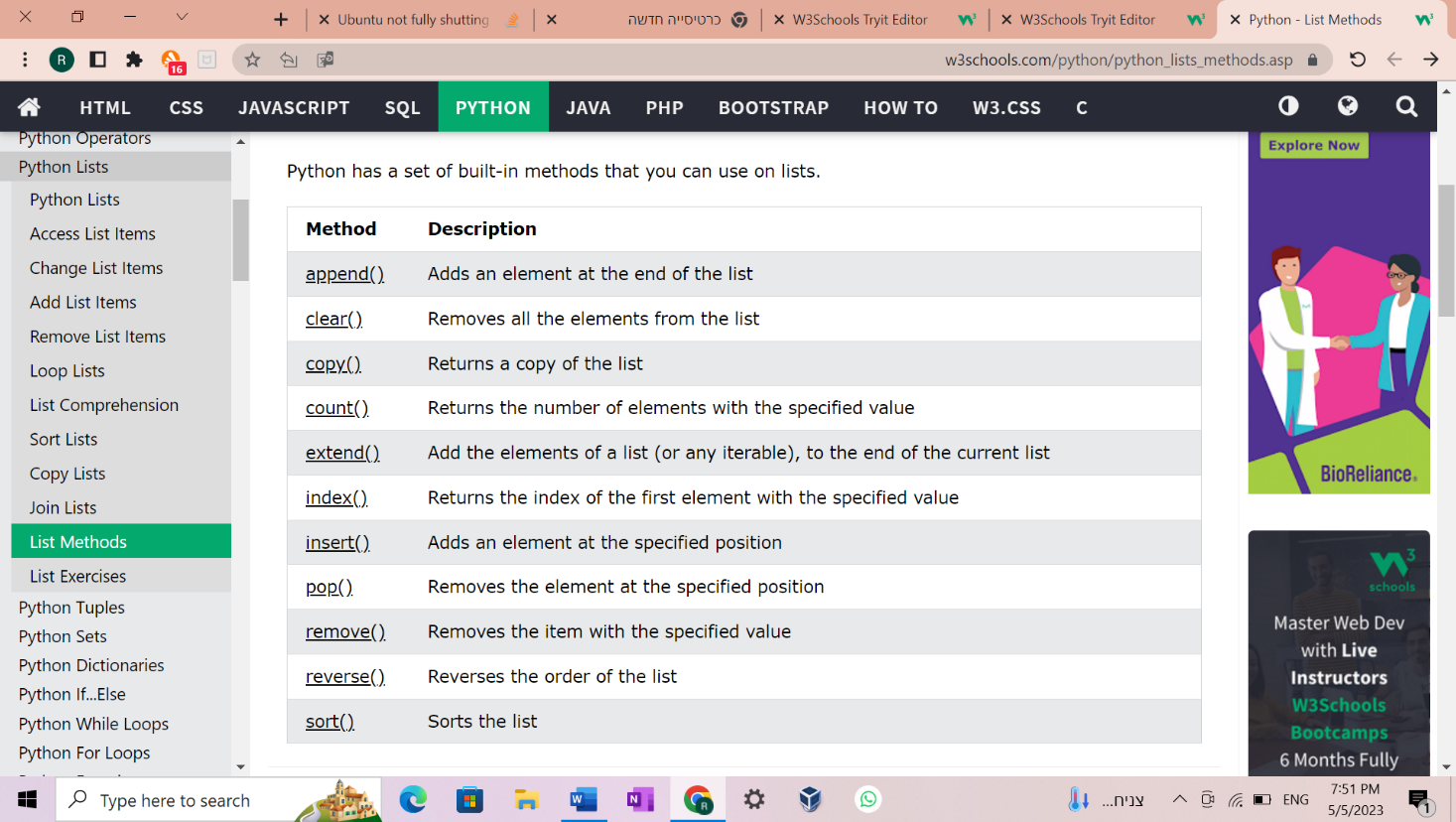
thislist = ["apple", "banana", "cherry"]

tropical = ["mango", "pineapple", "papaya"]

thislist.extend(tropical)

print(thislist)

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



**functions-**

If the number of arguments is unknown, add a \* before the parameter name:

Def mufun (\*num)

The following example shows how to use a **default parameter value**.

If we call the function without argument, it uses the default value:

def my\_function(**country = "Norway"**):

my\_function()

Search the string to see if it starts with "The" and ends with "Spain":

import re  
  
txt = "The rain in Spain"  
x = re.search("^The.\*Spain$", txt)

Python allows you to assign values to multiple variables in one line:

### x, y, z = "Orange", "Banana", "Cherry"

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

fruits = ["apple", "banana", "cherry"]  
x, y, z = fruits

**string-**

a = "Hello, World!"

print(a[0])

prints = H

Check if "expensive" is NOT present in the following text:

txt = "The best things in life are free!"  
print("expensive" not in txt)

To add a space between them, add a " ":

a = "Hello"  
b = "World"  
c = a + " " + b  
print(c)

Hello World

Use the format() method to insert numbers into strings:

age = 36  
txt = "My name is John, and I am {}"  
print(txt.format(age))

count() method –

Returns the number of times a specified value occurs in a string

txt = "I love apples, apple are my favorite fruit"

x = txt.count("apple")

print(x)

find() method- Searches the string for a specified value and returns the position of where it was found

txt = "Hello, welcome to my world."  
  
x = txt.find("welcome")  
  
print(x)

“7”

**Bool –**

Almost any value is evaluated to True if it has some sort of content.

Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

Set-

myset = {"apple", "banana", "cherry"}

Set items are unchangeable, but you can remove items and add new items.

Set items are unordered, unchangeable, and do not allow duplicate values.

## What is a Module?

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

You can summon modules with the command import

You can create a script, “my\_script” and include functions in it, like “fun1”.

Then you can “import my\_script” to where ever you want and use those functions.

When using a function from a module, use the syntax: *my\_script.fun1()*

1. Code Commit - This step involves writing and committing code changes to a version control system (VCS), such as Git. To get started, you can create a free account on GitHub or GitLab and start working on a project or contributing to an open-source project. Learn how to create a repository, clone it, make changes, and push the changes back to the repository. It's important to follow best practices for committing code, such as writing meaningful commit messages, committing small changes frequently, and keeping the code base clean and maintainable.
2. Continuous Integration (CI) - This step involves automating the build, test, and verification process of your code changes. To set up a CI pipeline, you can use tools like Jenkins, Travis CI, or CircleCI. You'll need to configure the pipeline to run automated tests, such as unit tests, integration tests, and acceptance tests, and ensure that your code changes integrate seamlessly with the rest of the codebase. You can start by setting up a simple pipeline that runs basic tests and gradually add more complexity as you gain experience.
3. Continuous Deployment (CD) - This step involves automating the deployment process of your code changes to a staging environment, where they can be tested and validated by stakeholders. To set up a CD pipeline, you can use tools like Ansible, Chef, or Puppet. You'll need to configure the pipeline to deploy the code changes automatically to the staging environment, run smoke tests, and notify stakeholders of any issues. It's important to ensure that the pipeline is reliable, resilient, and can handle failures gracefully.
4. Release - This step involves releasing the code changes to production, where they become available to end-users. To release code changes, you can use tools like Kubernetes or Docker Swarm. You'll need to ensure that the release process is safe, fast, and can be rolled back if necessary. It's important to have a robust release management process in place that involves thorough testing, documentation, and communication with stakeholders.

Files-

The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

The open() function returns a file object, which has a read() method for reading the content of the file:

f = open("demofile.txt", "r")  
print(f.read())

Open a file on a different location:

f = open("D:\\myfiles\welcome.txt", "r")  
print(f.read())

You can return one line by using the readline() method

f = open("demofile3.txt", "w")  
f.write("Woops! I have deleted the content!")  
f.close()

import os  
if os.path.exists("demofile.txt"):  
  os.remove("demofile.txt")  
else:  
  print("The file does not exist")

Remove the folder "myfolder":

import os  
os.rmdir("myfolder")

* [**List**](https://www.w3schools.com/python/python_lists.asp) is a collection which is ordered and changeable. Allows duplicate members. []
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) is a collection which is ordered and unchangeable. Allows duplicate members. ()
* [**Set**](https://www.w3schools.com/python/python_sets.asp) is a collection which is unordered, unchangeable\*, and unindexed. No duplicate members. {}
* **Dictionary** is a collection which is ordered\*\* and changeable. No duplicate members. { : }

in Python. To employ this module named “sys” is used. sys.argv is similar to an array and the values are also retrieved like Python array.

The sys module provides functions and variables used to manipulate different parts of the Python runtime environment. This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.

**import** sys

print("This is the name of the program:", sys.argv[0])

**print**("Argument List:", str(sys.argv))

* **len()-** function is used to count the number of arguments passed to the command line. Since the iteration starts with 0, it also counts the name of the program as one argument. If one just wants to deal with other inputs they can use (len(sys.argv)-1).
* **str()-** this function is used to present the array as a string array. Makes displaying the command line array easier and better.

The join() method takes all items in an iterable and joins them into one string.

A string must be specified as the separator.

myDict = {"name": "John", "country": "Norway"}  
mySeparator = "TEST"  
  
x = mySeparator.join(myDict)  
  
print(x)

nameTESTcountry

he split() method splits a string into a list.

You can specify the separator, default separator is any whitespace.

txt = "welcome to the jungle"  
  
x = txt.split()

x[0] = welcome

Split the string into a list with max 2 items:

txt = "apple#banana#cherry#orange"  
  
# setting the maxsplit parameter to 1, will return a list with 2 elements!  
x = txt.split("#", 1)  
  
print(x)

function return one argument !

so the way to overwrite it is to return an type of a list\tuple\set\dictionary.

Arbitrary Arguments, \*args

If you do not know how many arguments that will be passed into your function, add a \* before the parameter name in the function definition.

This way the function will receive a *tuple* of arguments, and can access the items accordingly:

Example

If the number of arguments is unknown, add a \* before the parameter name:

def my\_function(\*kids):  
  print("The youngest child is " + kids[2])  
  
my\_function("Emil", "Tobias", "Linus")

filter--> works only on Boolean functions, and filters the content using the filter algorithm specified in the function

map--> Doesn't matter what it does.

Filter--> actually filters the content, and appends the values that match the Boolean function algorithm. so only the values that match are stored.map-->EVERYTHING is stored, and it wont save the value, but save all the results and save it.

Lamda- nameless function

print([ 2 \* k for k in range(5)])

print([ k for k in range(0, 10, 2)])

print(list(k for k in range(10) if k % 2 == 0))

print(list(filter(lambda x : x % 2 == 0 , range(10))))

print(list(map(lambda x: 2\*x , range(5))))

[0, 2, 4, 6, 8]

[0, 2, 4, 6, 8]

[0, 2, 4, 6, 8]

[0, 2, 4, 6, 8]

[0, 2, 4, 6, 8]