* **List** 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 and unindexed. No duplicate members.
* [**Dictionary**](https://www.w3schools.com/python/python_dictionaries.asp) is a collection which is ordered\* and changeable. No duplicate members.

**List**: (Ordered, Changeable, Allows Duplicate)

list1 =[1, “Grapes”, True, “Flower”, “Mango”] or list2 = list((“Banana”, “Cherry”, “Melon”))

* List1**[1]**  🡪 1
* List1**[-1]** 🡪 Mango
* List1(**len**(list1)) 🡪 4
* list1**.append**(“Apple”) 🡪 [1, “Grapes”, True, “Flower”, “Mango”, “Apple”]
* list1**.insert**(2, “Marigold”) 🡪 [1, “Grapes”, ”Marigold”, True, “Flower”, “Mango”, “Apple”]
* list1**.pop**() 🡪 [1, “Grapes”, ”Marigold”, True, “Flower”, “Mango”]
* list1**.pop(2)** 🡪 Marigold
* list1**.remove**(“True”) 🡪 [1, “Grapes”, ”Marigold”, “Flower”, “Mango”, “Apple”]
* **del** list1**[2]** 🡪 [1, “Grapes”, “Flower”, “Mango”, “Apple”]
* list1**.reverse**() 🡪 [“Apple”, “Mango”, “Flower”, “Grapes”, 1]
* list2 = **sorted(**list1) 🡪 [“Apple”, “Flower”, “Grapes”, “Mango”]
* list**[2]** = “kiwi” 🡪 [“Apple”, “kiwi”, “Grapes”, “Mango”] **#replace**
* list**[1:3] =** “Blackberry”,”Cherry” 🡪 [“Apple”, “Blackberry”, ”Cherry”, “Mango”]

(works with string list or separate integer list can’t use in a str and int list)

* list1**.clear()** 🡪 []
* **del** list1 🡪 will completely remove list1
* newlist = **[0]\*5** 🡪 [0, 0, 0, 0, 0]
* **JOIN TWO** List: newlist = [0, 0, 0, 0, 0] newlist1 = [1,2,3,4,5,6]
  + - newlist2 = newlist **+** newlist1 🡪 [0,0,0,0,0,1,2,3,4,5,6]
    - for x in newlist1:

newlist.append(x) 🡪 [0,0,0,0,0,1,2,3,4,5,6]

* + - newlist.extend(newlist1) 🡪 [0,0,0,0,0,1,2,3,4,5,6]
* **List Slicing:**

A = newlist**[2:6]** 🡪 [0,0,0,1]

B = newlist**[:2]** 🡪 [0,0,0]

C = newlist**[1::2]** 🡪 [0,0,3,6]

* **COPY list:** listorg = [“apple”, “banana”, “cherry”]

list\_cpy  **=** listorg 🡪 [“apple”, “banana”, “cherry”] (original list will also updated if new list is updated)

newlist = listorg**.copy()** 🡪 [“apple”, “banana”, “cherry”]

(originallist will not change if newlist is append with new item)

list\_cpy **=** listorg**[:] 🡪** [“apple”, “banana”, “cherry”] (original list will also updated if new list is updated)

* **Create New List from Existing List:**
* x = [“papaya”, “lemon”, “dates”] y = [“papaya”, “pineapple”] z = (“papaya”, “pineapple”)

x**.extend(y)** 🡪 [“papaya”, “lemon”, “dates”, [“papaya”, “pineapple”]]

x**.extend(y)** 🡪 [“papaya”, “lemon”, “dates”, (“papaya”, “pineapple”)]

* **for i in x:**

**print(i) 🡪** papaya lemon dates

* **for x in range(len(x)):**

**print(x)** 🡪 0 1 2

* **a = 0**

**while a < len(x):**

**print(x[a])**

**a = a + 1 🡪** papaya lemon dates

* **print((p) for p in x) 🡪** papaya lemon dates
* Syntax: newlist = [expression for item in iterable if condition == True]
* **newlist = []**

**for m in x:**

**if “a” in m:**

**newlist.append(x)**

**print(newlist) 🡪 [ ‘**papaya‘, ‘dates’ ]

* **listnew = [m for m in x if “a” in m]🡪 [** ‘papaya’ , ‘dates’ **]**
* listnew1 = **[s for s in x if s != “papaya”]** 🡪 [ ‘lemon’ , ‘dates’ ]
* listnew2 = **[t for t in x] 🡪** [‘papaya’, ‘leamon’, ‘dates’]
* listnew3 = **[x for x in range(10)]**
* **RANGE()** in List:

listrng = [**j for j in range(**8**)**] 🡪 [0, 1, 2, 3, 4, 5, 6, 7]

listrng = [**h for h in range(**10**) if h < 5**] 🡪 [0, 1, 2, 3, 4]

* **UPPER()** in List: car = [“toyota”, “Honda”]

lstupr = [**x.upper() for x in car**] 🡪 [“TOYOTA”, “HONDA”]

* **SET all values**:

alvalu = [**‘hello’ for k in car**] 🡪 [‘hello’, ‘hello’]

* **REPLACE all same values:** fruiti = **[**"apple", "banana", "cherry", "kiwi", "mango", “banana”**]**

newlist = **[u if u !=** “banana**” else** “orange**” for x in** fruiti]

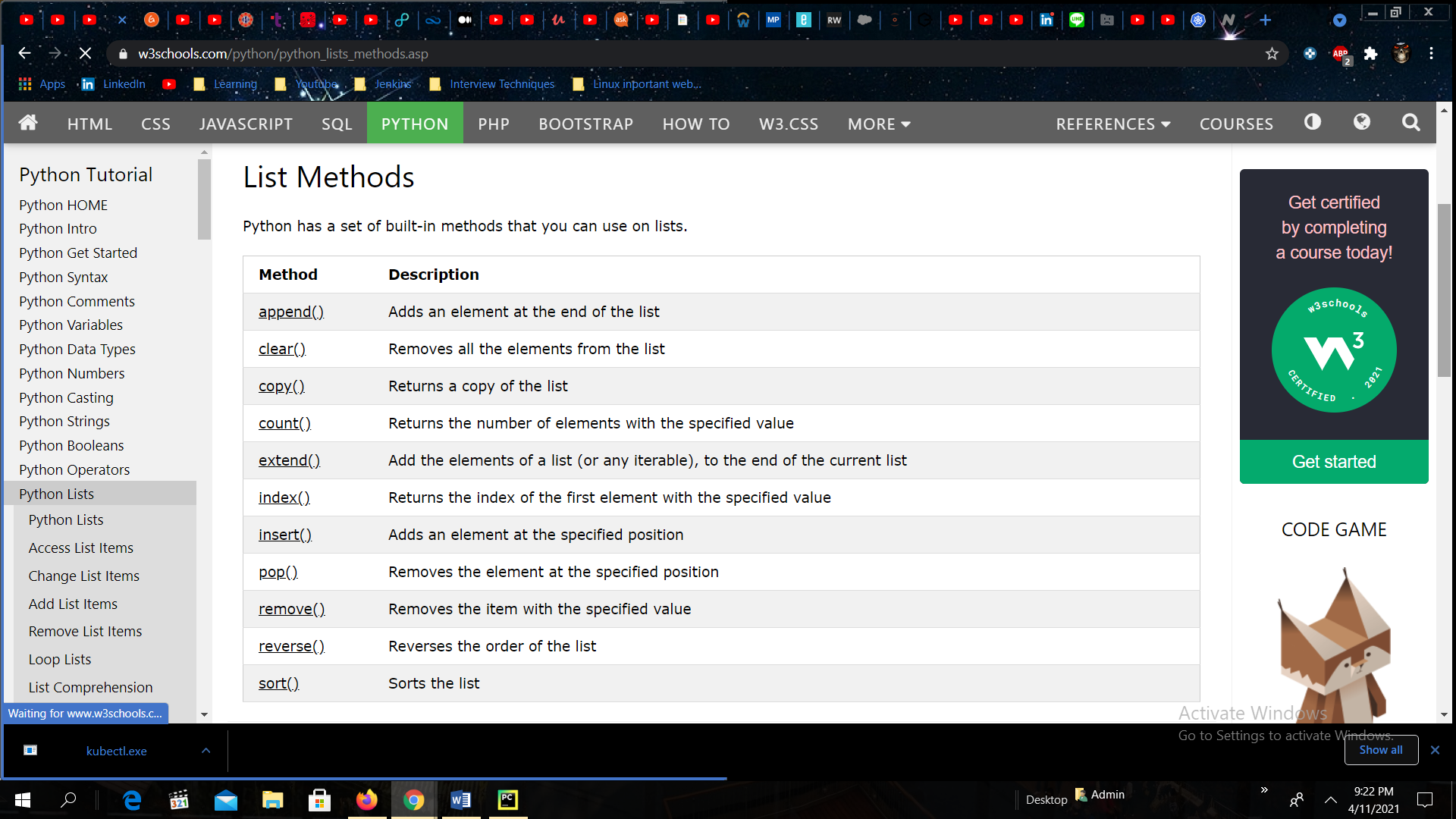
🡪[“apple”, “orange”, “cherry”, “kiwi”, “mango”, “orange”]

* SORT: thislist = ["orange", "banana", "mango", "kiwi", "pineapple", "banana"] thislist1=[100,50,65,82,23]

thislist**.sort()** 🡪 [“banana”, ”banana”, “kiwi”, “mango”, “orange”, “pineapple”]

thislist1**.sort()** 🡪 [23,50,65,82,100]

thisllist.sort(reverse = True) 🡪 [“pineapple” , “orange”, “mango”, “kiwi”, “banana”, ”banana”]

* **Sqr**list = **[i\*i** **for i in newlist1**]
* 

**TUPLE**

Collection of data item which is immutable, and **can not be changed** after creation, ordered, **UNCHANGABLE(.append cannot be used)**

Mytuple = (“Max”) 🡪 type 🡪 str

Mytuple = (“Max”,) 🡪 type 🡪 tuple

mytuple = (“Max”, 28, “Boston”)

* Tuple from list:

Mytuple1 = **tuple(**[“Max”, 28, “Boston”]**)** 🡪 (“Max”, 28, “Boston”, 28, 28) 🡨 type tuple

* Item = mytuple1[2] 🡪 Boston
* Item = mytuple1[-2] 🡪 28
* **If 28 in** Mytuple1:

print(“Yes, 28”)

**else:**

print(“NO”) 🡪 Yes, 28

* print(**len(**Mytuple1)) 🡪 5
* print(Mytuple1**.count(**28)) 🡪 3
* print(Mytuple**.index(“**Boston”) 🡪 2
* **SLICING:** print(Mytuple**[2:5])** 🡪 (“Boston”, 28, 28)

o = print(Mytuple**[::2])** 🡪 (“Max”, “Boston”, 28)

print(Mytuple**[::-1])** 🡪 (28, 28, “Boston”, 28, “Max”)

* **REMOVE:** **a** = (“apple”, “banana”, “cherry”, “strawberry”, “rasberry”)

a**.remove(**“apple”**)** 🡪 (“banana”, “cherry” , “strawberry”, “rasberry”)

**del** a 🡪 print(a) 🡪 #this will raise an error because the tuple no longer exists

* **Asterisk \*** :

**(**first, second, **\*third) = a**

**Print(first) 🡪** apple

**Print(second) 🡪** banana

**Print(third) 🡪 [**‘cherry’, ‘strawberry’, ‘rasberry’**]**

**(**first, **\*second,** third, fourth **) = a**

**Print(first) 🡪** apple

**Print(second) 🡪 [‘**banana’, ‘ cherry’]

**Print(third) 🡪** strawberry

**Print(fourth) 🡪** raspberry

* **LOOP:** b = (“one”, “two”, “three”, “four”)
  + - * + **for x in a:** print(a) 🡪 one two three four
        + **for i in range(len(a)):** print(**i**) 🡪 0 1 2 3 4
        + **for i in range(len(a)):** print(**a[i])** 🡪apple banana cherry strawberry raspberry
        + **i = 0**

**while i < len(a):**

**print(a[i])**

**i = i + 1** 🡪 apple banana cherry strawberry raspberry

* **Multiple Tuple:**

**My\_tuple = b \* 2 🡪** (“one”, “two”, “three”, “four”, “one”, “two”, “three”, “four”)

name, city, age = o 🡪 print(name) 🡪 Max

tuple will take less memory and less time to execute as compared to List.

==========================================================================================

**DICTIONARIES**

Key-value pairs, unordered, mutable

mydict = {“name”: “Max”, “age”: 28, “city”:”Boston”}

* mydict2 = dict(name = “Mary”, age = 25, city = “New York”)
* value = mydict2[“city”] 🡪 New York
* mydict**[“email”]** = [abc@ghj.com](mailto:abc@ghj.com) 🡪 print(mydict) 🡪 {“name”: “Max”, “age”: 28, “city”:”Boston”, “email” : “abc@ghj.com”}
* **del** mydict[“age”] 🡪 {“name”: “Max” , “city”:”Boston”, “email” : “abc@ghj.com”}
* mydict**.pop**(“city”) 🡪 {“name”: “Max”, “email” : “abc@ghj.com”}
* mydict**.popitem()** 🡪 {“name”: “Max” }
* if “name” in mydict:

print(mydict[“name”]) 🡪 Max

* for key in mydict:

print(key) 🡪 name age city email

* print(mydict.keys()) 🡪 name age city email
* print(mydict.value()) 🡪 Max 28 Boston [abc@ghj.com](mailto:abc@ghj.com)
* value = mydict[3] 🡪
* mydict1 = {“name”: “Max”, “age” : 28, “city”: “Boston”, “email”:”abc@ghj.com”}

mydict2 = dict(name = “Marry”, age = 25, city = “New York”)

mydict1.update(mydict2) 🡪 { “name” = “Marry”, “age” = 25, “city” = “New York”, “email”:”abc@ghj.com”}

* #Convert twolist into dictionary: list = ["Naina", "Kimi", "Sheena"] list1 = [852345,763567,691276]

def list\_dict():

out = **dict(zip**(list, list1**))**

print(out)

list\_dict() 🡪 {'Naina': 852345, 'Sheena': 691276, 'Kimi': 763567}

* #Convert Dictionary to Tuple:

def dic\_to\_tupl():

x = {'Naina': 852345, 'Sheena': 691276, 'Kimi': 763567}

for m in x.items():

print(m)

dict\_to\_tupl() 🡪 ('Naina', 852345) ('Sheena', 691276) ('Kimi', 763567)

* tuple can be converted to dictionary but list cannot be converted to dictionary

-------------------------------------------------------------------------------------------------------------------------------------------------------------------

**SETS**

**(Unordered, Unchangeable, Duplicated NOT Allowed)**

* mysets = {1,2,3,4,5} 🡪 {1,2,3,4,5}

mysets = {1,2,3,4,5,1,1,2,3} 🡪 {1,2,3,4,5}

print(len(mysets)) 🡪 5

* myset = set([1,2,3,4,5,2,3,6]) 🡪 {1,2,3,4,5,6}

myset = set(“Hello”) 🡪 H e l o

* mysets**.add**(8) 🡪 {1,2,3,4,5, 8}

mysets**.add**(9) 🡪 {1,2,3,4,5,8,9}

mysets**.add**(7) 🡪 {1,2,3,4,5,8,9,7}

* mysets**.remove**(3) 🡪 {1,2,4,5,8,9,7} (if the item to remove does not exist, remove() will raise an error)
* mysets**.discard**(7) 🡪 {1,2,4,5,8,9}

mysets**.discard**(10) 🡪 even 10 is not present it will not give error

* mysets**.clear**() 🡪 {}
* mysets**.pop()** 🡪 {2,4,5,8,9} 🡪 will delete first item
* **Union and Intersecion (**will not change original , will always return output**)**
* odds = {1,3,5,7,9} even = {0,2,4,6,8} primes = {2,3,5,7}
* u = odds**.union**(even) 🡪 {0,1,2,3,4,5,6,7,8,9}
* I = odds**.intersection**(evens) 🡪 set{} 🡪intersection will give same item

I = odds**.intersection**(primes) 🡪 {3,5,7}

I = even**.intersection**(primes) 🡪 {2}

* Calculate difference of two sets: setA = {1,2,3,4,5,6,7,8,9} setB = {1,2,3,10,11,12}

setC = {“banana”, “cherry”, “mango”} setD = [“Leamon”, “Orange”]

* diff = setA**.diference**(setB) 🡪 {4,5,6,7,8,9}

diff = setB**.difference**(setA) 🡪 {10,11,12}

* diff = setB**.symmetric\_difference**(setA)🡪 {4,5,6,7,8,9,10,11,12}

diff = setA**.symmetric\_difference**(setB)🡪 {4,5,6,7,8,9,10,11,12}

* setA**.update**(setB) 🡪 {1,2,3,4,5,6,7,8,9,10,11,12}

setC.update(setD) 🡪 {“banana”, “cherry”, “mango”, “Leamon”, “Orange”}

* setA**.intersection\_update**(setB) 🡪 {1,2,3}
* setA**.difference\_update**(setB) 🡪 {4,5,6,7,8,9} 🡪it will remove same items from setA
* setA**.symmetric\_difference\_update(**setB**)** 🡪 {4,5,6,7,8,9,10,11,12}
* setA = {1,2,3,4,5,6}
* setB = {1,2,3}
* print(setA**.issubset**(setB)) 🡪 False (subset means all the items of setA are in setB which is False)
* Now setAA = {1,2,3} setBB = {1,2,3} setC = {7,8,9}

Print(setAA**.issubset**(setBB)) 🡪 True

* Print(setA**.issuperset**(setB)) 🡪 False 🡪(superset means that all the items from setA is in set B which is False)

Print(setB**.issuperset**(setA)) 🡪 True

* Print(setA**.isdisjoint**(setB)) 🡪 False 🡪(disjoint means both the sets have diff. items)
* Print(seta**.isdisjoint**(setC)) 🡪 True
* Access List: thiset = {“apple”, “banana”, “cherry”}
* for x in thiset: print(x) 🡪 apple banana cherry
* print(“banana” in thiset) 🡪 True

=====================================================================================================

**STRING**

**Strings are immutable(you can’t** change the item directly**)**

* my\_string = ‘hello World’ “Hello World” “’Hello Worls”’

my\_string = “Hello \

world” 🡪 Hello world

* char = my\_string[2:5] 🡪 llo
* my\_string[0] = ‘h’ 🡪 Error does not support item assignment
* my\_new\_string = my\_string[2:8] 🡪 llo Wo 🡪(8th is excluded) **#String SLICING**

my\_new\_string = my\_string[2:] 🡪 llo World

my\_new\_string = my\_string[::2] 🡪 HloWrd

my\_new\_string = my\_string[::-1] 🡪 dlroW olleH

* greeting = “Hello” name = “Tom” **#CONCATINATE**

sentence = greeting + “ “ + name 🡪 Hello Tom

* for i in greeting:

print(i) 🡪H e l l o

* if “ell” in greeting: print(“Yes e”) if “lle” in greeting: print(“Yes”)

else: print(“NO”) 🡪 Yes e else: print(“NO”) 🡪 NO

* my\_strings = “ Hello World “ 🡪 Hello World

my\_strings = my\_strings**.strip()** 🡪 Hello World **#REMOVE WHITESPACE**

* my\_string**.upper()** 🡪 HELLO WORLD **#UPPER and LOWER**

my\_string**.lower()** 🡪 hello world

* print(my\_string**.stratswith(**“H”)**)** 🡪 True

print(my\_string**.stratswith(‘**World’)**)**🡪 False

* print(my\_string**.endswith(**“World”)**)** 🡪 True

prin(my\_string**.endswith(**“orl”)**)** 🡪 False

* Return Index:

Print(my\_string**.find**(‘o’)) 🡪 4

Print(my\_string**.find**(‘wo’)) 🡪 6

Print(my\_string**.find**(‘p’))) 🡪 -1

* Count no of Strings:

My\_string**.count**(‘l’) 🡪 2

My\_string**.count**(‘p’) 🡪 0

* My\_string**.replace(‘**world’, ‘universe’**)** 🡪 Hello universe **#REPLACE**

(this will return new string and will not change the old original string).

* **Convert STRING to LIST:**

My\_string = ‘how are you doing?’ my\_string1 = ‘how,are,you,doing’

My\_list = My\_string.split() 🡪 [‘how’, ‘are’, ‘you’, ‘doing’] 🡪 .split(“ ”) default argument is space

My\_list = my\_string1(“,”) 🡪 [‘how’, ‘are’, ‘you’, ‘doing’]

* **Converting List to String:**

my\_list = [‘how’, ‘are’, ‘you’, ‘doing’]

new\_string = ‘ ‘.join(my\_list) 🡪 howareyoudoing

new\_string1 = ‘ ‘.join(my\_list) 🡪 how are you doing

my\_list = [‘a’] \* 6 🡪 [‘a’ ,’a’ ,’a’, ’a’, ’a’, ’a’ ]

or

my\_string = ‘’ # BAD APPROACH

for i in my\_list:

my\_string += 1 🡪 aaaaaa

#Good Approach my\_string = ‘’.join(my\_list) 🡪 aaaaaa

* **% Place holder:**

**FOR STRING:** var = “Tom” var1 = “Timm”

my\_string = “this variable is %s “ %var + “ tis for %s” %var1 🡪 this variable is Tom this for Timm

**FOR Decimal** var = 8 var1 = 7

my\_string = “this is for %d “ %var + “ this is for second %d “ %var1 🡪 this is for 8 this is for second 7

**For Float:** flot = 3.2254 flot1 = 2.0159

my\_floaty = “this is a first flot %f” %flot + “ this for second flot %flot1” %flot1 🡪 this is a first flot 3.2254 this is for second flot 2.0159

* Quantity = 6 itemno = 586 price = 596.325 **#FORMAT**

Order = “I want {} pieces of item{} for {}dollars.”**.format(**Quantity, itemno, price**)**

🡪 I want 6 pieces of item 586 for 596.325 dollars.

Order1 = “I want to pay {0} for item no {1} for {2} quantity.”.format(price, itemno, quantity)

🡪 I want to pay 596.325 for item no 586 for 6 quantity.

* newtxt = “we are the so called\” DamPeoples \” from heaven. ” **#ESCAPE CHARACTER**
* we are the so called “DamPeoples” from heaven.
* txt = “the best thing is the life are free!”

print(“free” **in** txt) 🡪 True

* if “thing” **in** txt:

print(“Yes thing is present”) 🡪 Yes thing is present

* print(“hi” **not in** txt) 🡪 False
* if “batball” **not in** txt:

print(“No BatBall”) 🡪 NO Batball

BOOLEAN

🡪 Returns TRUE or FALSE. Any List, Tuple, Set and Dictionary are True, except empty ones.

* Print(10> 9) 🡪 True

Print(10 < 9) 🡪 False

Print(10 == 9) 🡪 False

* Print(bool(“Hello”)) 🡪 True

Print(bool(568) 🡪 True

Print(bool([“apple”, “cherry”, “banana”]) 🡪 True

* x = “Hello” y = 45

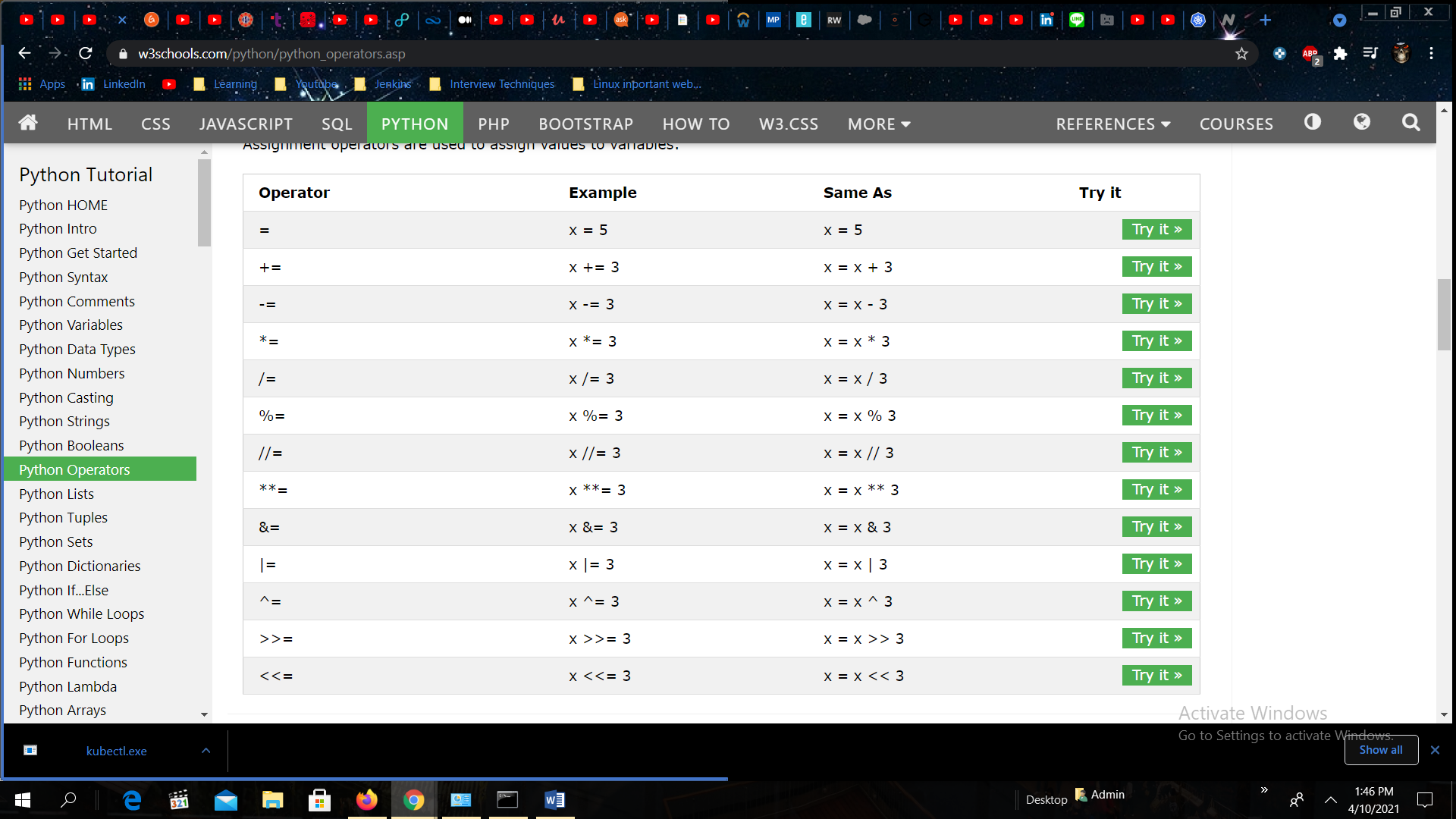
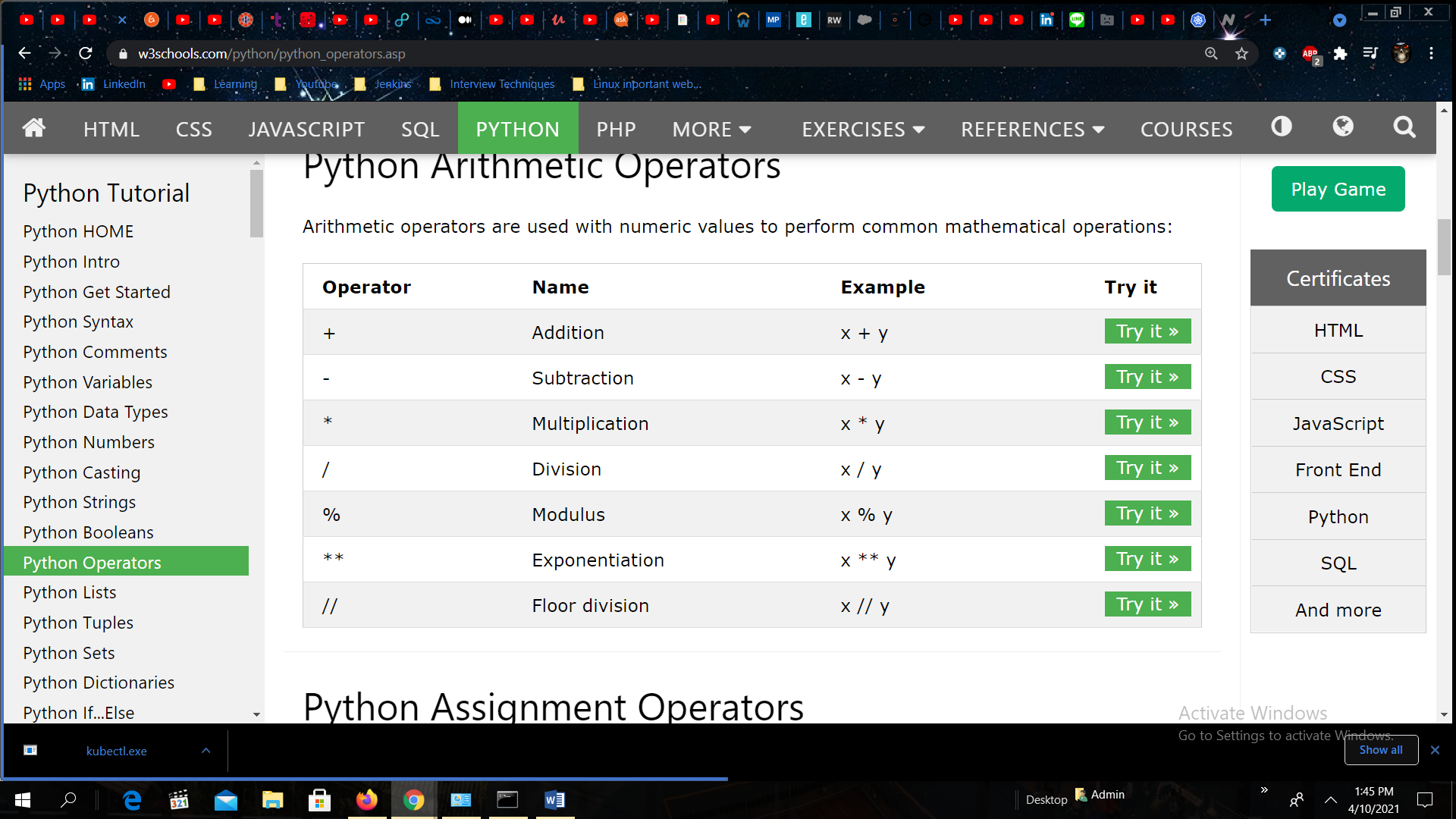
print(bool(x)) 🡪 True

print(bool(y)) 🡪 True

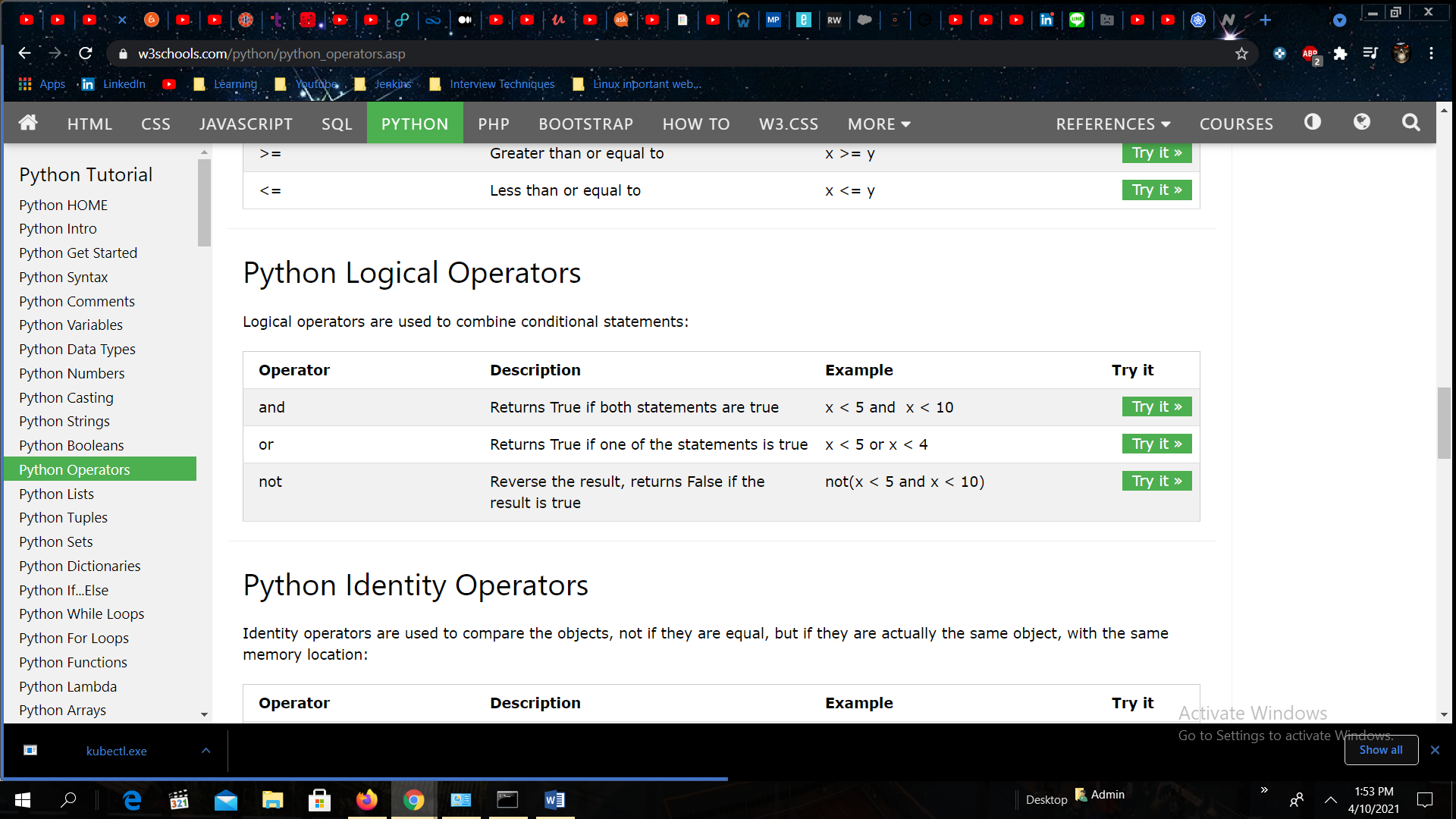
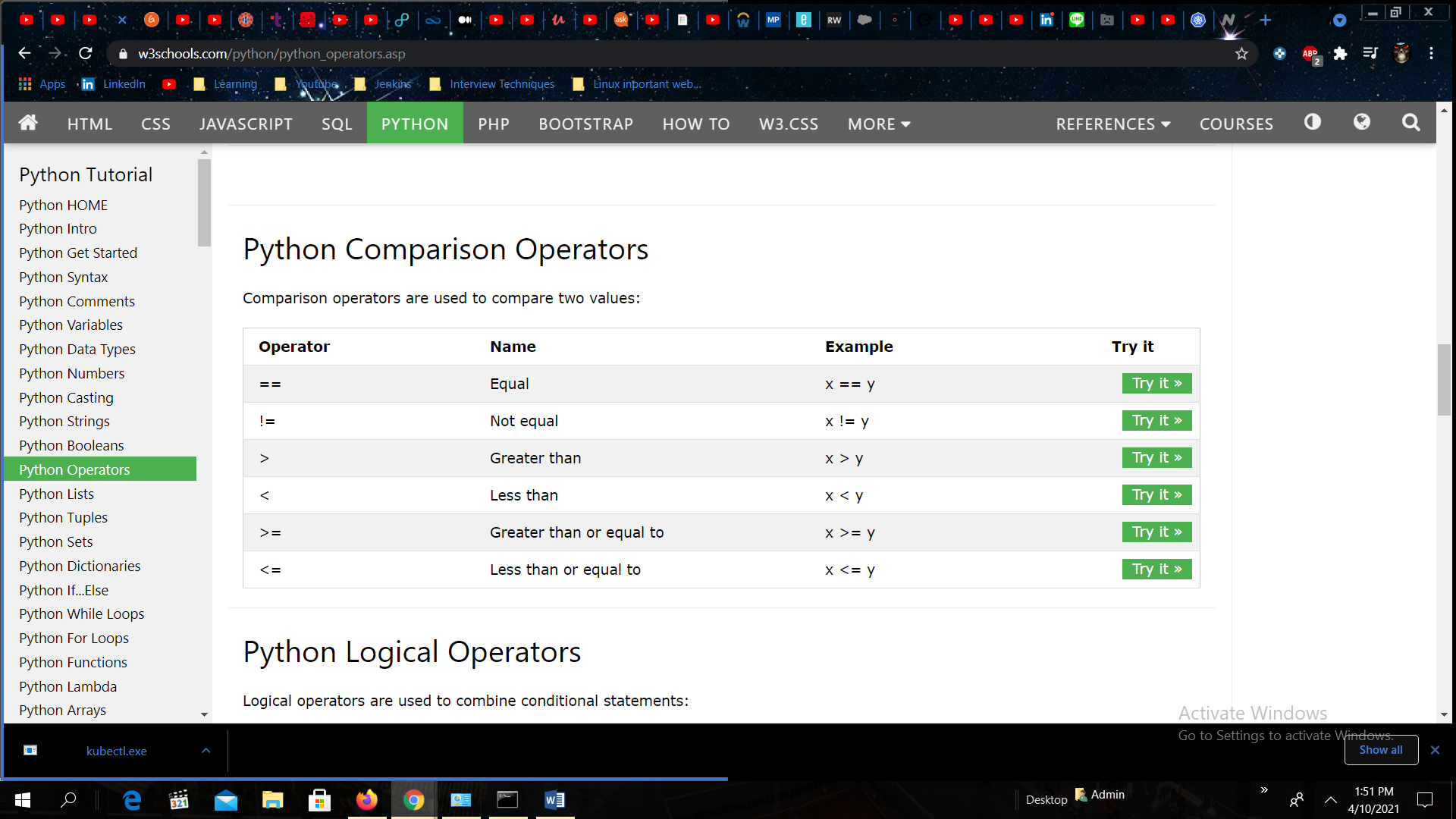
print(bool(False)) 🡪 False

**OPERATORS**

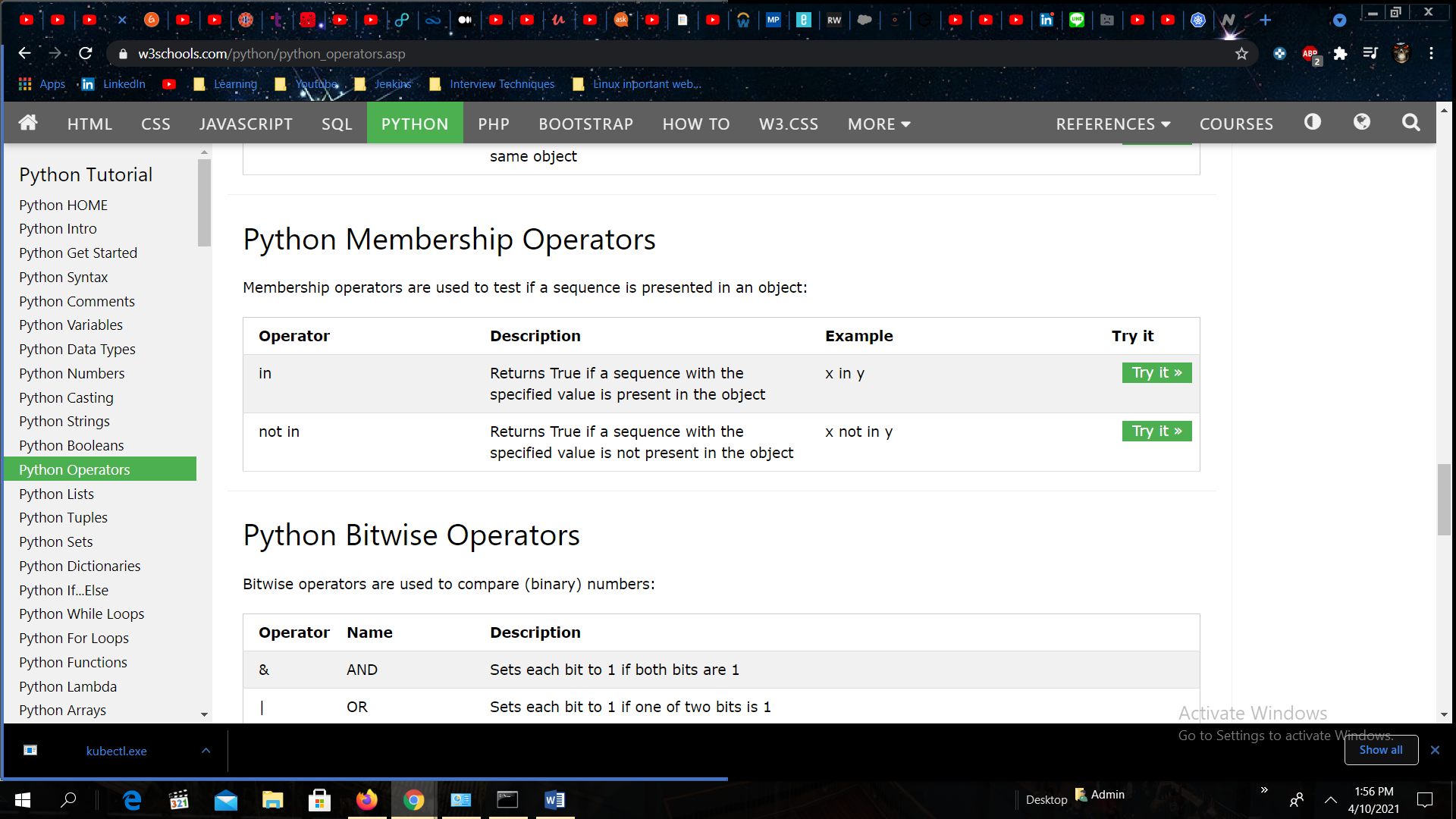
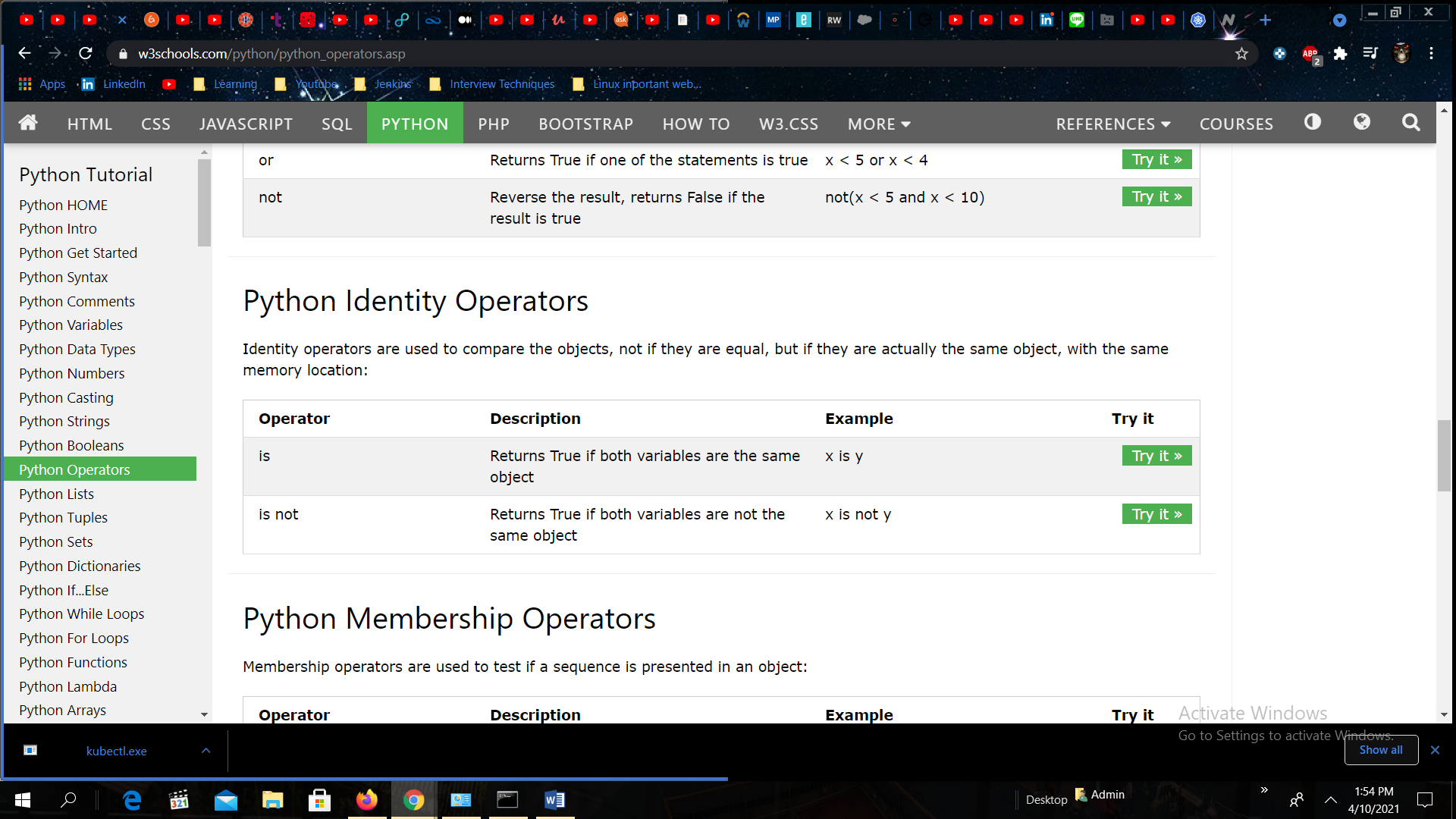
ARITHMETIC Operators ASSIGNMENT Operator



COMPARISION Operator LOGICAL Operator



IDENTITY Operator MEMBERSHIP Operator



BITWISE Operator

