SEQUENCES

They can be divided into two categories based on the ordering of items: **Sequences** and **Collections**. Elements in sequences **come out** in the **same order** as it is **inserted**, however **ordering** in collections is **not preserved**. In simple words collections are not in sorted form and that cannot be sorted even manually they keeps changing their position.

Python Sequences.

In Python programming, sequences are a **generic term** for an **ordered set** which means that the order in which we **input** the items will be the same when we **access** them. Python supports **six** different types of sequences. These are

- **lists**
- **Dictionary**
- **tuples**
- **byte sequences**
- **byte arrays**
- **range objects**

• lists:

Python lists are similar to an array but they allow us to **create** a **heterogeneous collection** of items inside a **list**. A list can contain **numbers**, **strings**, **lists**, **tuples**, **dictionaries**, **objects**, etc.

Lists are declared by using **square brackets** around **comma-separated** items.

Syntax:

```
list1 = [1,2,3]
list2 = ['red',6,7, 'green', 'blue']     (list has both numbers and string)
list3 = ['hello', 100, 3.14, [1,2,3]]   (list in list)
```

(Note: variable must not be named as 'list' it can be list1, a, b, list2)

list = [1,2,3,'hello',7,8] ✗ list1=[1,2,3,'hello',7,8] ✓

because name 'list' is inbuilt function used for type conversion

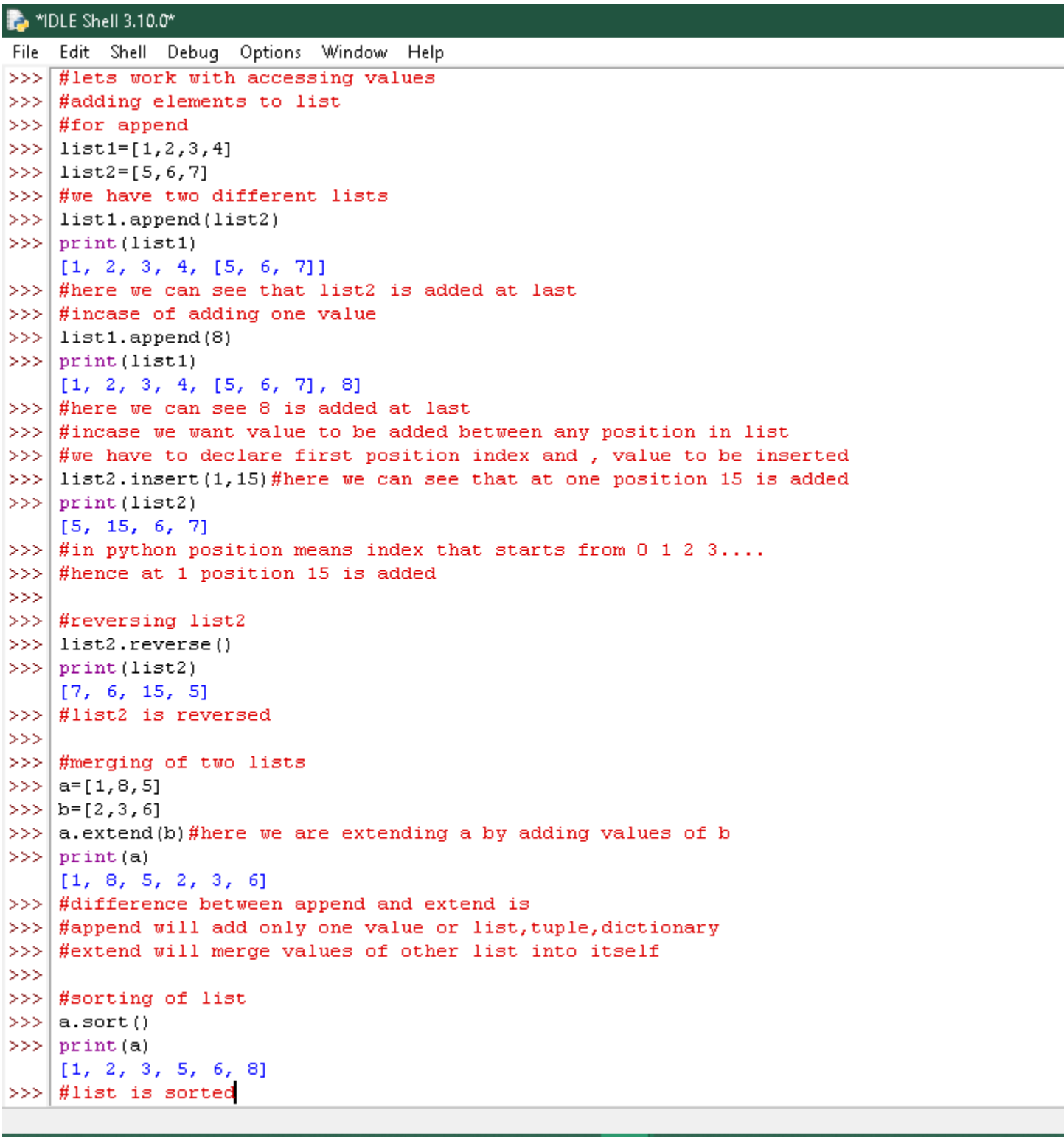
Lists are **mutable** which makes it easier to **change** and we can quickly **modify** a list by directly **accessing** it.

Python List Methods	
Method	**Description**
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

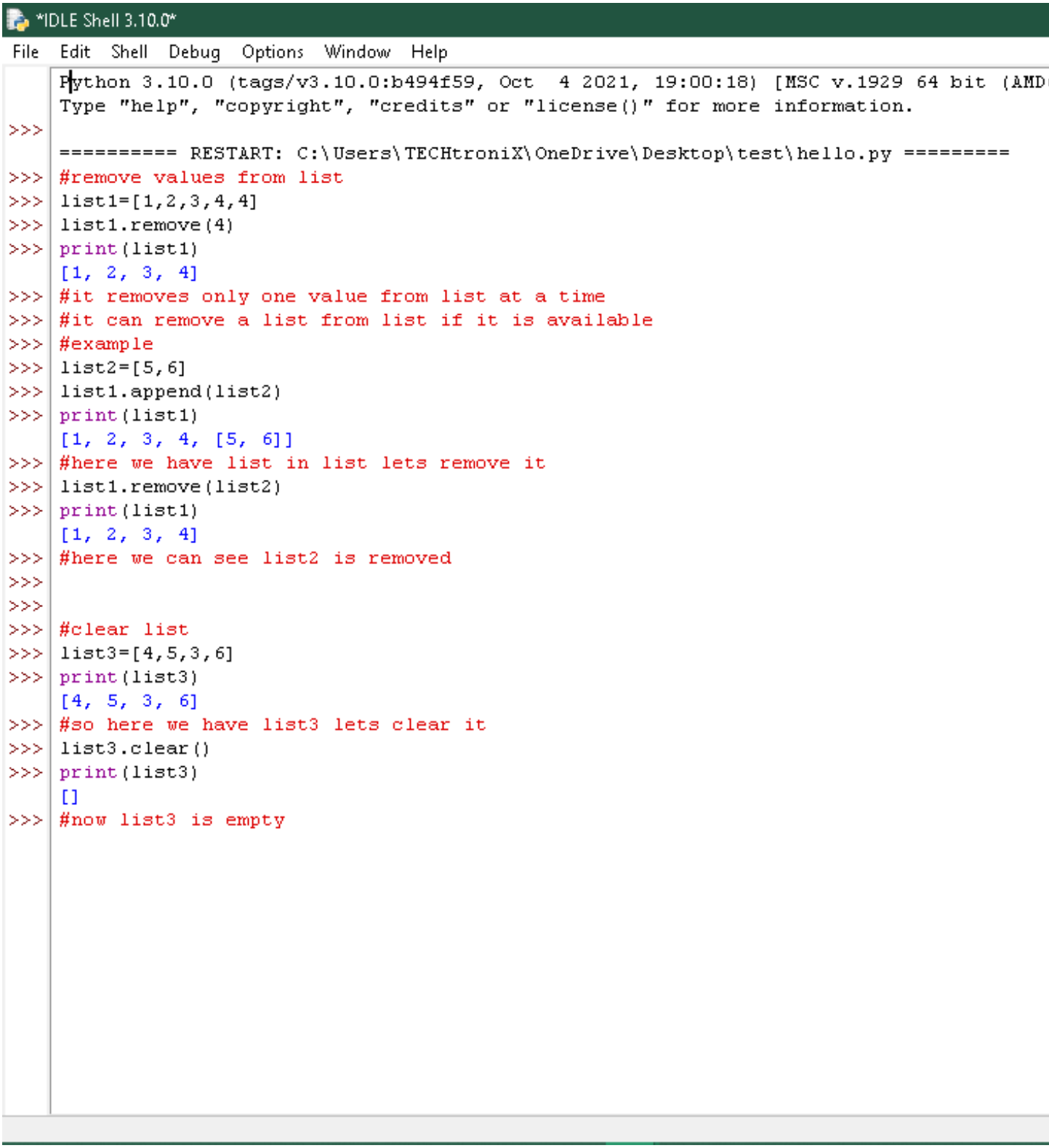
Syntax:

```
Listname.append(value)

Listname.clear()

List1.extend(list2)

Listname.insert(position,value)

Listname.pop(position)

Listname.remove(value)

Listname.sort()

Listname.reverse()
```

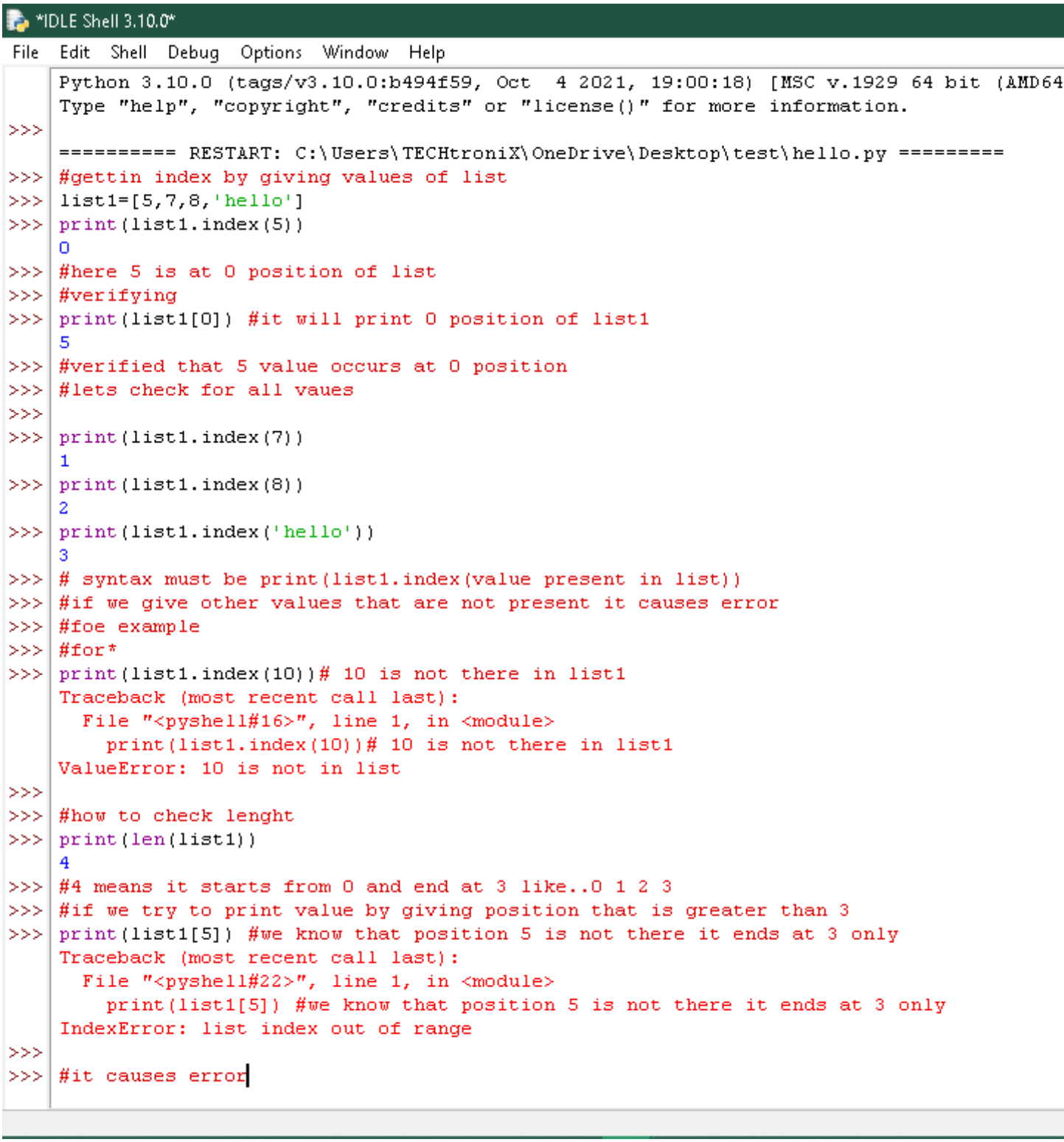
Adding, extending and sorting by giving values of list



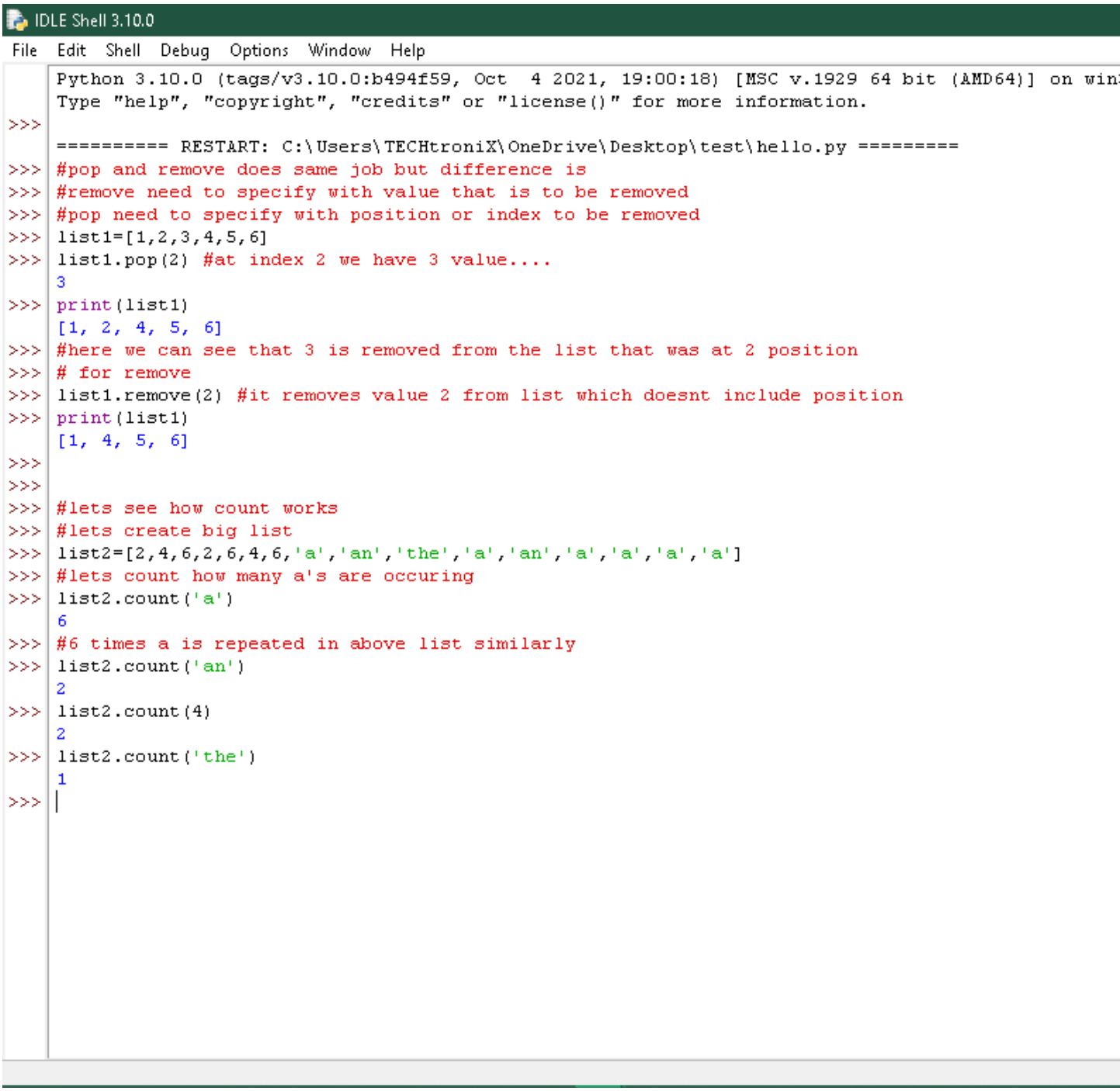
Removing and clearing list by values



Knowing index (position) of values in the list



Pop and count



- **Dictionaries:**

Python dictionaries are similar to list that it is mutable they allow us to **create** a **heterogeneous collection** of items inside a **dictionary**. Dictionary is a set of key and value that are enclosed by {} curly braces

Syntax:

{key1: value, key2: values}

Values can be of any type but key cannot be Boolean (true or false)

It is different from list in accordance of operations as index is not available in dictionaries

List is operated by index and values whereas dictionary has specified each key to each value manually

Operations	Example	Description

Creating a dictionary	>>> a={1:"one",2:"two"}
	>>> print(a)
	{1: 'one', 2: 'two'}
		Creating the dictionary with elements of different data types.

accessing an element	>>> a[1]
	'one'
	>>> a[0]
	KeyError: 0
		Accessing the elements by using keys.

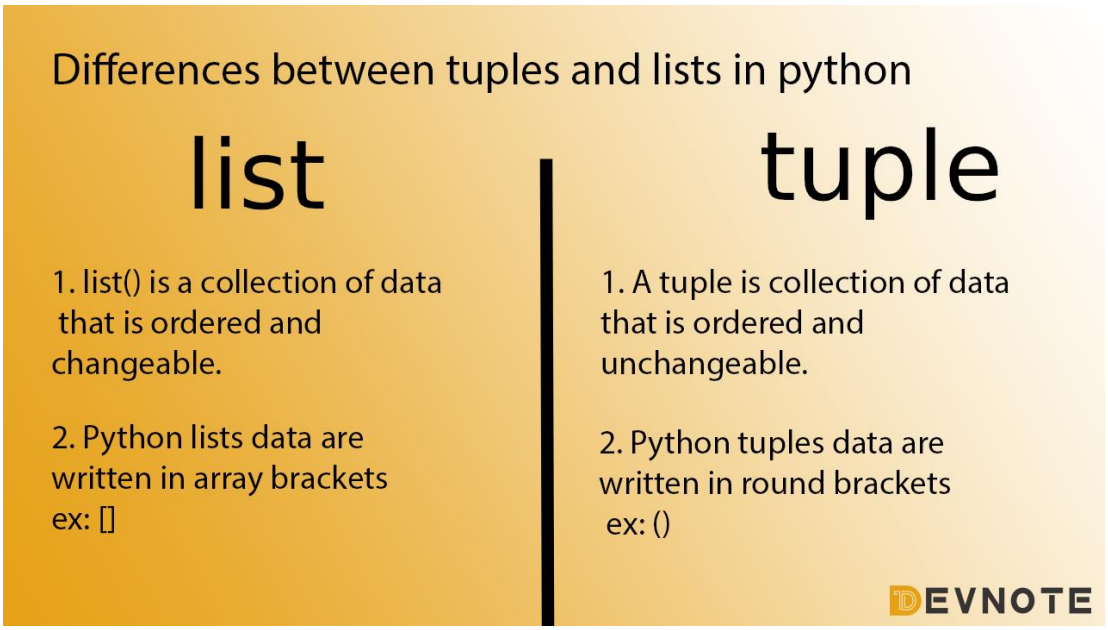
Update	>>> a[1]="ONE"
	>>> print(a)
	{1: 'ONE', 2: 'two'}
		Assigning a new value to key. It replaces the old value by new value.

add element	>>> a[3]="three"
	>>> print(a)
	{1: 'ONE', 2: 'two', 3: 'three'}
		Add new element in to the dictionary with key.

membership	a={1: 'ONE', 2: 'two', 3: 'three'}
	>>> 1 in a
	True
	>>> 3 not in a
	False
		Returns True if the key is present in dictionary. Otherwise returns false.

Tuples:

Tuples is same as list it consists of heterogeneous elements in it but it cannot be changed or extended as list does Difference between list and tuples.



Syntax:

tuple1= (1,2,3,4,5,'hello',9)

print(type(tuple1)

print(tuple1[3])

output:

<class 'tuple'>
4

Note: we cannot add or remove particular element or cannot find index of values as it is immutable.

Concatenation of Tuples

```
# Code for concatenating 2 tuples

tuple1 = (0, 1, 2, 3)
tuple2 = ('python', 'geek')

# Concatenating above two
print(tuple1 + tuple2)
```

Output:

```
(0, 1, 2, 3, 'python', 'geek')
```

Nesting of Tuples

```
# Code for creating nested tuples

tuple1 = (0, 1, 2, 3)
tuple2 = ('python', 'geek')
tuple3 = (tuple1, tuple2)
print(tuple3)
```

Output :

```
((0, 1, 2, 3), ('python', 'geek'))
```

Repetition in Tuples

```
# Code to create a tuple with repetition

tuple3 = ('python',)*3
print(tuple3)
```

Output

```
('python', 'python', 'python')
```

Slicing in Tuples

```
# code to test slicing

tuple1 = (0 ,1, 2, 3)
print(tuple1[1:])
print(tuple1[::-1])
print(tuple1[2:4])
```

Output

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

Deleting a Tuple

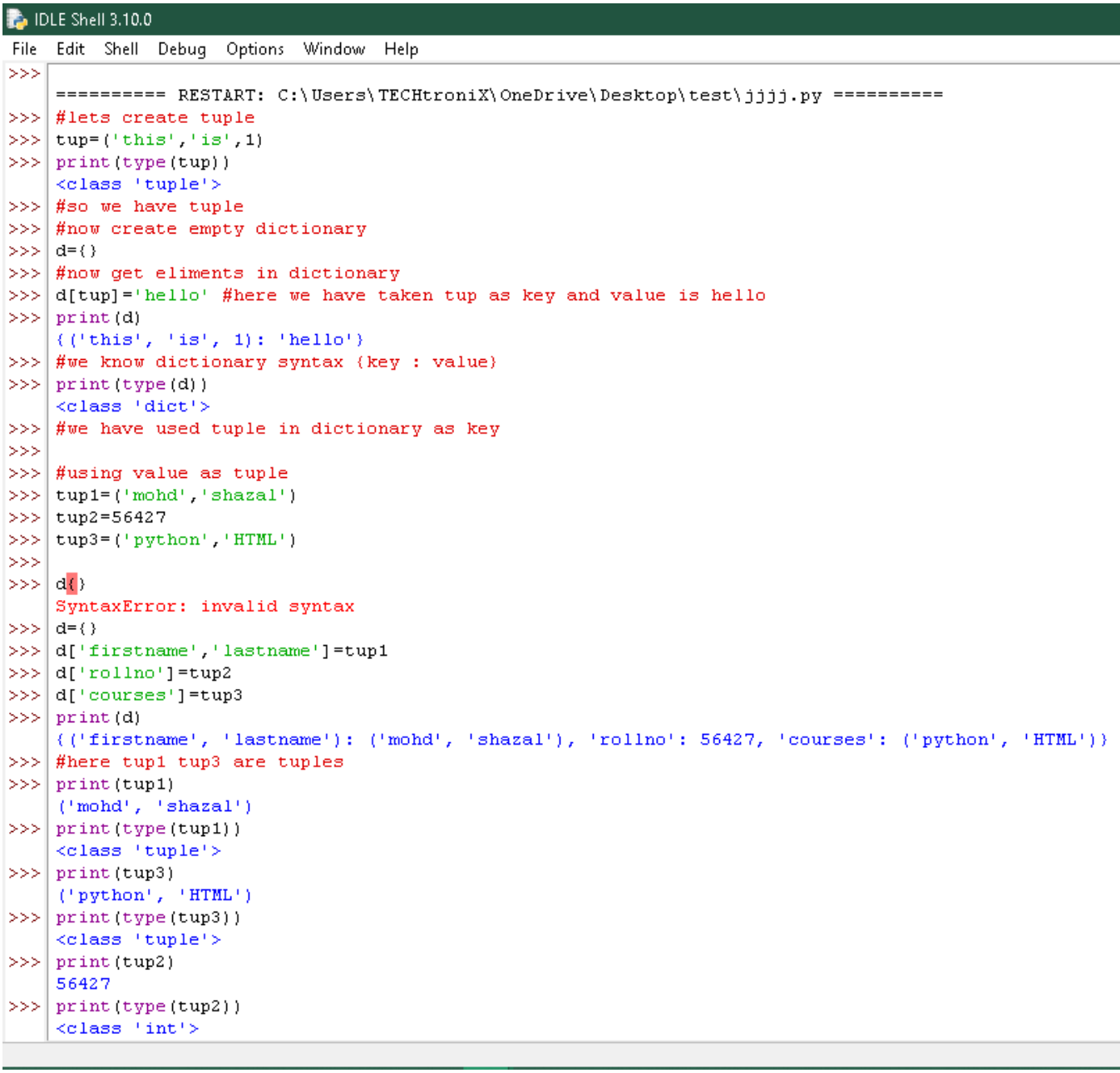
```
# Code for deleting a tuple

tuple3 = ( 0, 1)
del tuple3
print(tuple3)
```

Error:

```
Traceback (most recent call last):
  File "d92694727db1dc9118a5250bf04dafbd.py", line 6, in <module>
    print(tuple3)
NameError: name 'tuple3' is not defined
```

Tuples can be used in dictionaries as key or as values

- First create tuples then implement it in dictionary by creating empty dictionary as **variable={}**


IDLE Shell 3.10.0
File Edit Shell Debug Options Window Help
>>>
========== RESTART: C:\Users\TECHtroniX\OneDrive\Desktop\test\jjjj.py ==========
>>> #lets create tuple
>>> tup=('this','is',1)
>>> print(type(tup))
<class 'tuple'>
>>> #so we have tuple
>>> #now create empty dictionary
>>> d={}
>>> #now get eliments in dictionary
>>> d[tup]='hello' #here we have taken tup as key and value is hello
>>> print(d)
{('this', 'is', 1): 'hello'}
>>> #we know dictionary syntax {key : value}
>>> print(type(d))
<class 'dict'>
>>> #we have used tuple in dictionary as key
>>>
>>> #using value as tuple
>>> tup1=('mohd','shazal')
>>> tup2=56427
>>> tup3=('python','HTML')
>>>
>>> d()
SyntaxError: invalid syntax
>>> d={}
>>> d['firstname','lastname']=tup1
>>> d['rollno']=tup2
>>> d['courses']=tup3
>>> print(d)
{('firstname', 'lastname'): ('mohd', 'shazal'), 'rollno': 56427, 'courses': ('python', 'HTML')}
>>> #here tup1 tup3 are tuples
>>> print(tup1)
('mohd', 'shazal')
>>> print(type(tup1))
<class 'tuple'>
>>> print(tup3)
('python', 'HTML')
>>> print(type(tup3))
<class 'tuple'>
>>> print(tup2)
56427
>>> print(type(tup2))
<class 'int'>

SETS (COLLECTION)

Sets are collection of elements where we can add elements to set at the end but cannot be removed or changed it does not have particular positions (index) to access

Set Example

```
set1 = {"Ram", "Arun", "Kiran"}

set2 = {16, 78, 32, 67}

set3 = {"apple", "mango", 16, "cherry", 3}
```

we can add tuple and list elements to sets….

adding list elements to set by using update()

```
set1 = {1, 2, 3, 4, 5}

# a list of numbers to add

list_to_add = [5, 6, 7]

# add all elements of list to the set

set1.update(list_to_add)

print('Updated set after adding elements: ', set1)
```

Output:

```
Updated set after adding elements: {1, 2, 3, 4, 5, 6, 7}

It does not same values twice as of 5 above repeated twice
```

Multiple lists or tuples can also be added

```
# input set

set1 = {11, 12, 13, 14}

# 3 lists of numbers

list1 = [15, 16, 17]

list2 = [18, 19]

list3 = [30, 31, 19, 17]

# Add multiple lists

set1.update(list1, list2, list3)

#updated list

print('Updated Set: ', set1)
```

Output:

```
Updated Set: {11, 12, 13, 14, 15, 16, 17, 18, 19, 30, 31}
```

Instead of **update()** we can use **|** this symbol

```
#original set

set1 = {1, 2, 3, 4, 5}
```

```
#list of numbers to add

list1 = [6, 7]

# convert list to set and get union of both the sets using |

set1 |= set(list1)



#updated set

print('Updated Set: ', set1)
```

Output:

```
Updated Set: {1, 2, 3, 4, 5, 6, 7}
```

By using add() we can add tuple to set but cannot add list

```
#input set

set1 = {1, 2, 4, 5}

# tuple to add

tuple1 = (6, 7)

#add tuple to the set

set1.add(tuple1)

#prints updated set

print("Updated set after adding tuple: ", set1)
```

Output:

```
Updated set after adding tuple: {1, 2, 4, 5, (6, 7)}
```

If we try to add list to set

```
#input set

set1 = {1, 2, 3, 4, 5}

#list of numbers to add

list1 = [6,7]

# add list to the set

set1.add(list1)

print('Updated set after adding element: ', set1)
```

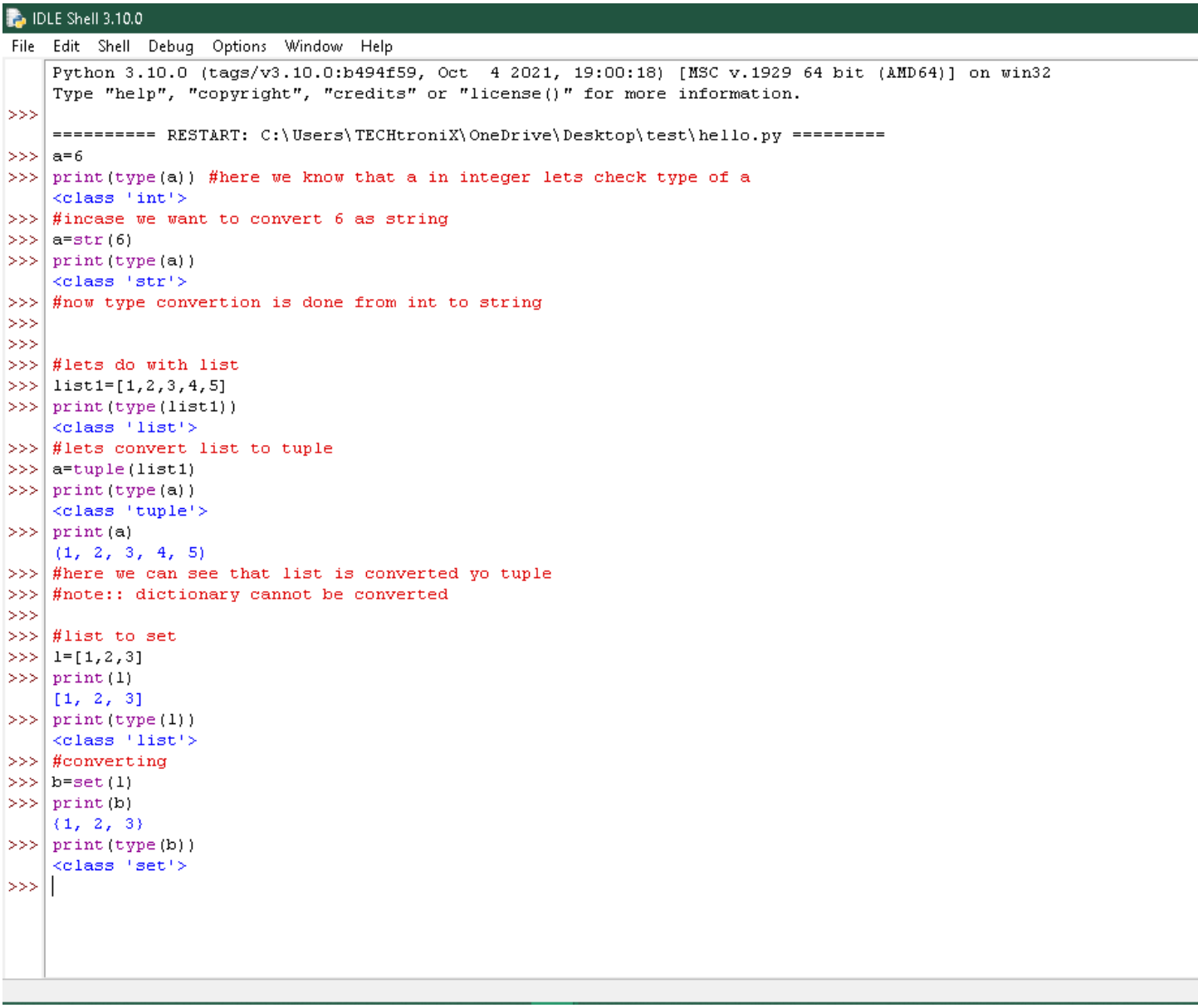
Output:

```
TypeError: unhashable type: 'list'
```

Type conversion:

converting one type to other type is called type conversion

For example:



Built-in Functions				
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	