Python Data Structures I: Strings, Lists, Tuples

Hina Arora

Python Data Structures

- There are four basic sequence types. Sequences are indexed by number.
  - **Strings** are immutable sequences of Unicode code points.
  - **Lists** are mutable sequences typically used to store collections of homogeneous items.
  - **Tuples** are immutable sequences typically used to store collections of heterogeneous data.
  - **Ranges** are immutable sequence of numbers typically used to loop through for loops.
- **Sets** are unordered collection with no duplicate elements. Sets can not be indexed.
- **Dictionaries** are unordered set of <key:value> pairs, where the keys are of immutable type, and must be unique within one dictionary. Dictionaries are indexed by keys.

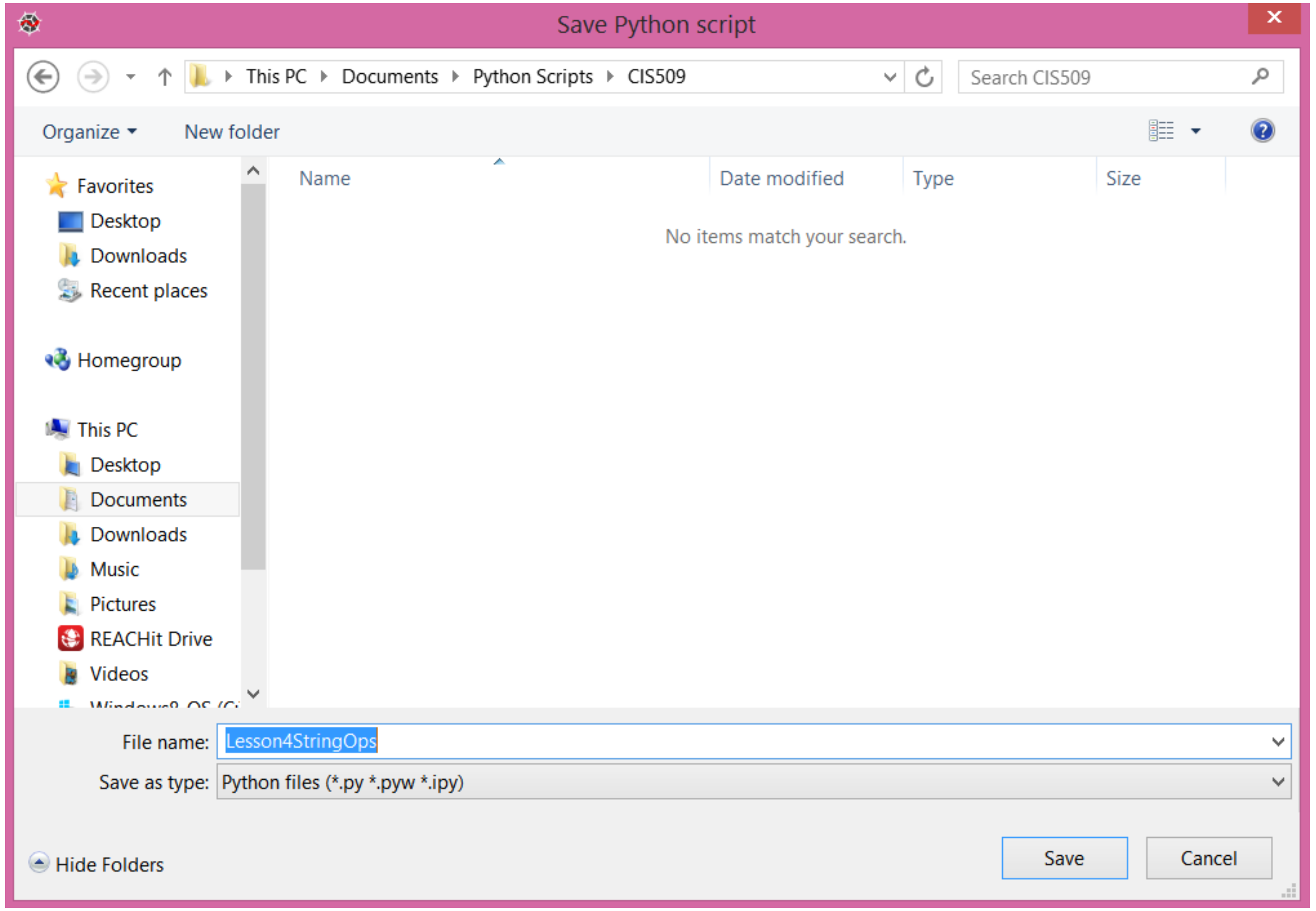
Strings and Operators

follow along!

(Reference: https://docs.python.org/3/tutorial/index.html)

Create a new file called “Lesson4StringOps.py”

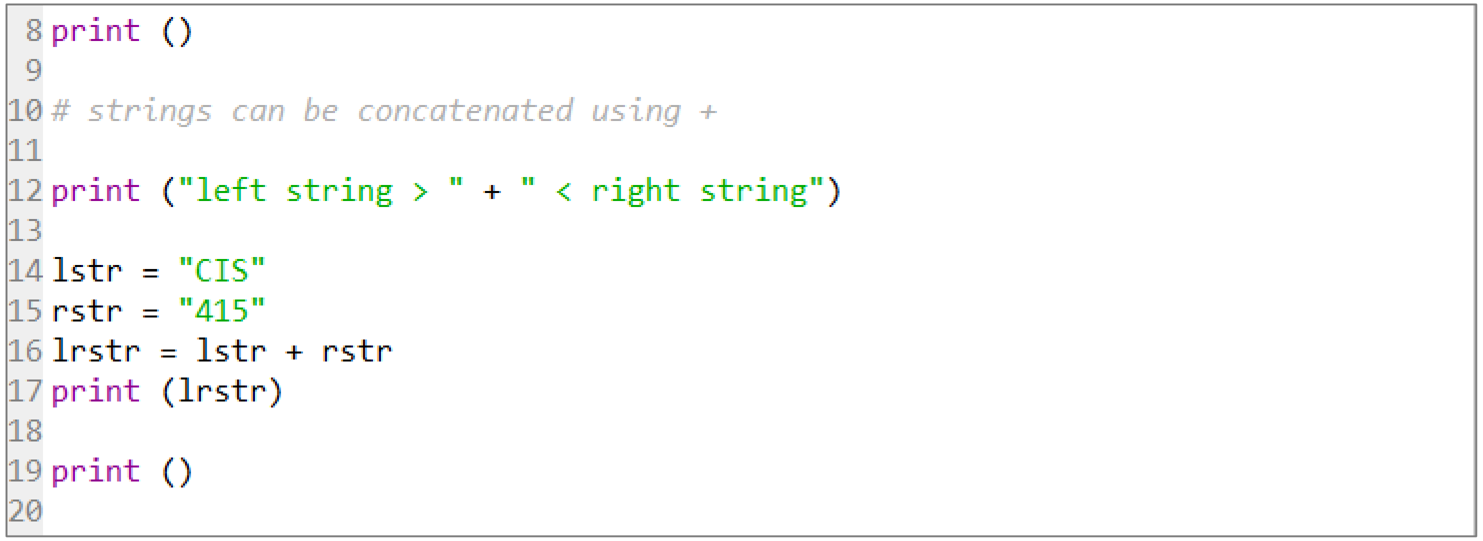
- Go to File -> New file…
- Go to File -> Save as…
- Go to CISPy directory
- Save file as “Lesson4StringOps.py”



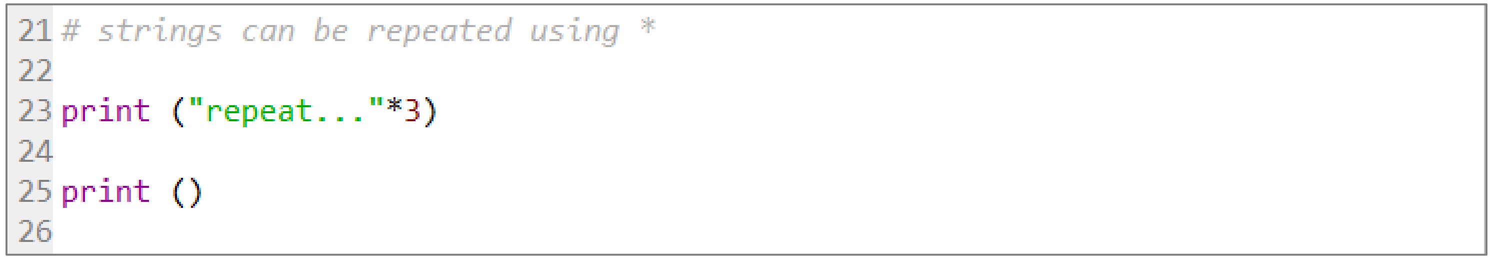
No items match your search.

File name: Lesson4StringOps

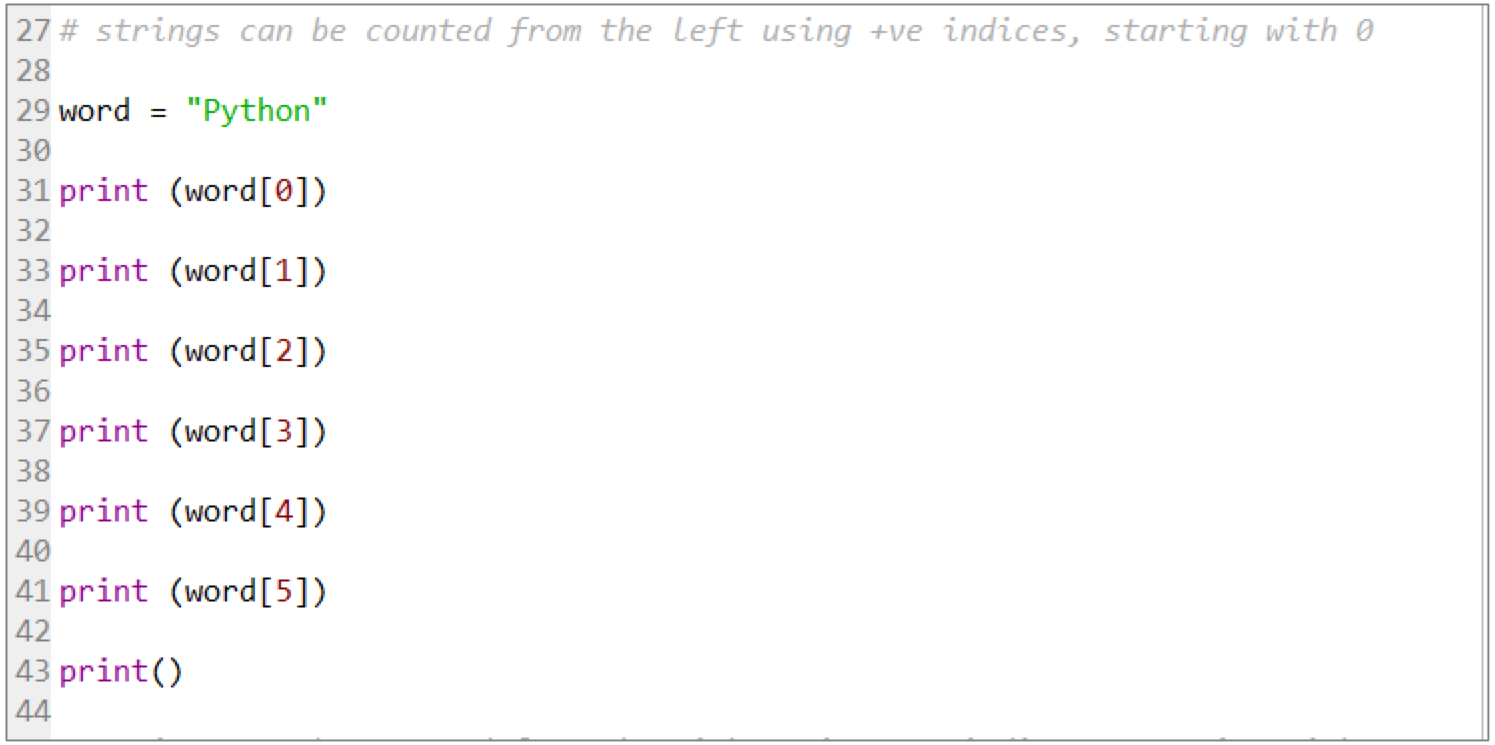
Save as type: Python files (*.py *.pyw *.ipy)



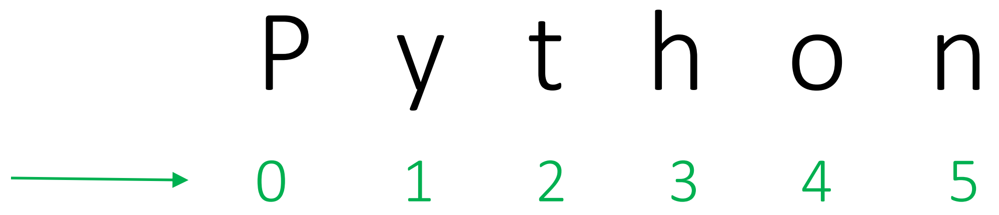
```python
print ()

# strings can be concatenated using +

print ("left string > " + " < right string")

lstr = "CIS"
rstr = "415"
lrstr = lstr + rstr
print (lrstr)

print ()

```

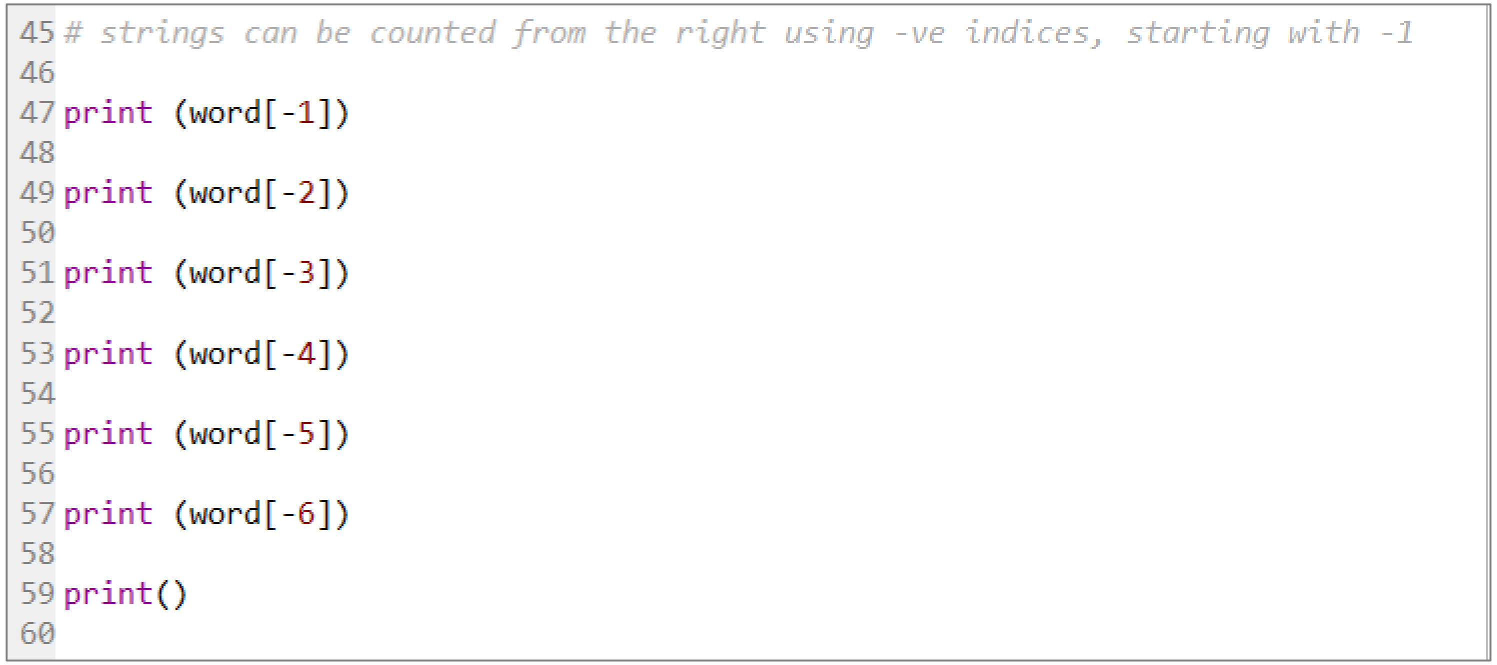


```python
# strings can be repeated using *

print ("repeat..."*3)

print ()

```

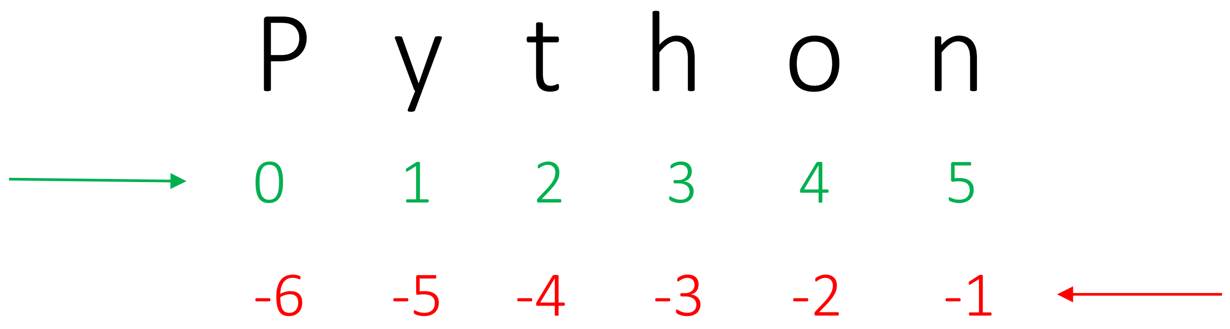


```python
# strings can be counted from the left using +ve indices, starting with 0

word = "Python"

print (word[0])

print (word[1])

print (word[2])

print (word[3])

print (word[4])

print (word[5])

print()

```





```python
# strings can be counted from the right using -ve indices, starting with -1

print (word[-1])

print (word[-2])

print (word[-3])

print (word[-4])

print (word[-5])

print (word[-6])

print()

```

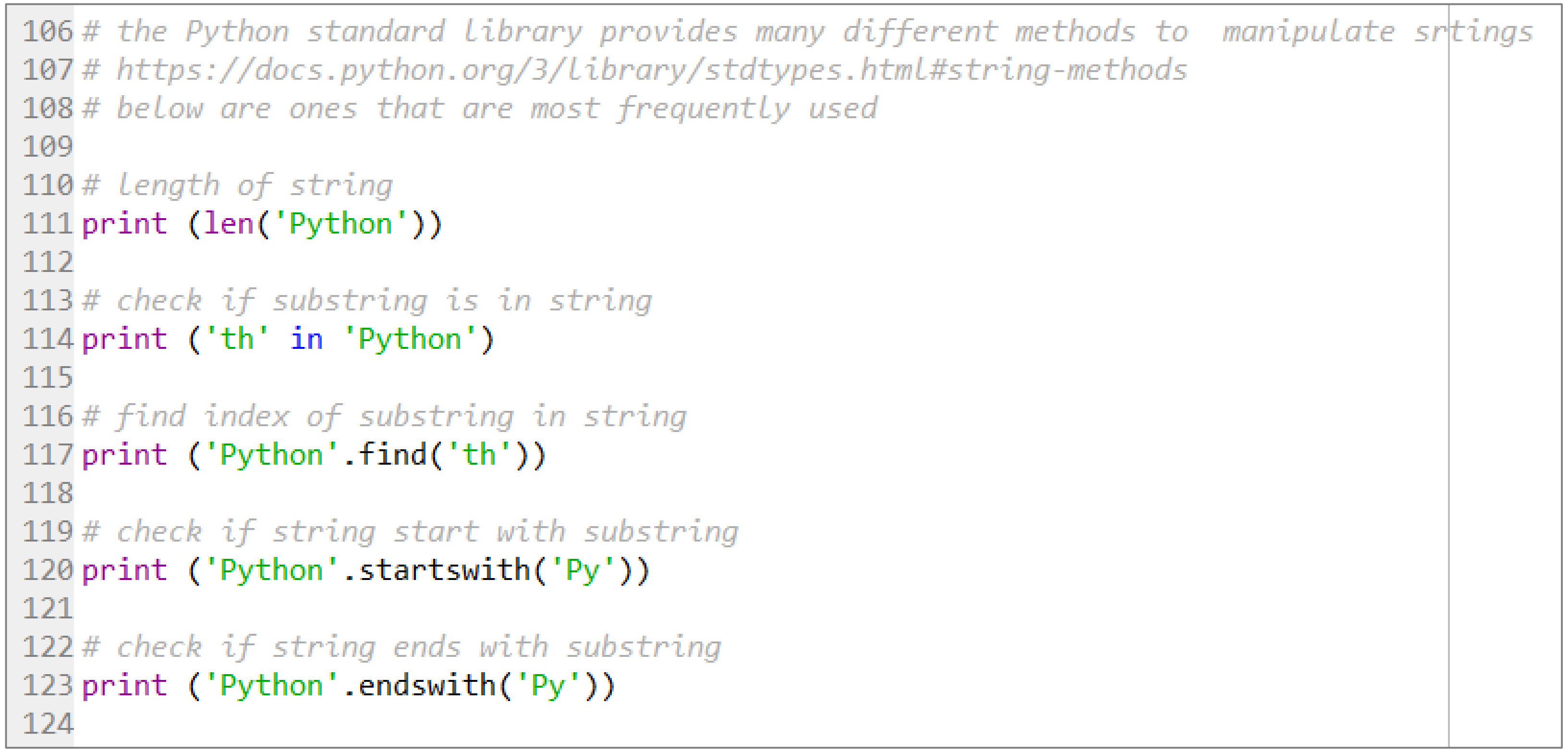


```python
# strings can be sliced with [startIndx:endIndx]:
# - startIndx is included and endIndx is excluded
# - startIndx must be < endIndx, else empty string is returned
# - if a slice index is out of range, python will go as far as it can

print (word[0:2])

print (word[2:6])

print (word[2:10])

print (word[-3:-2])

print (word[-3:-6])

print (word[-3:6])

print ()

```

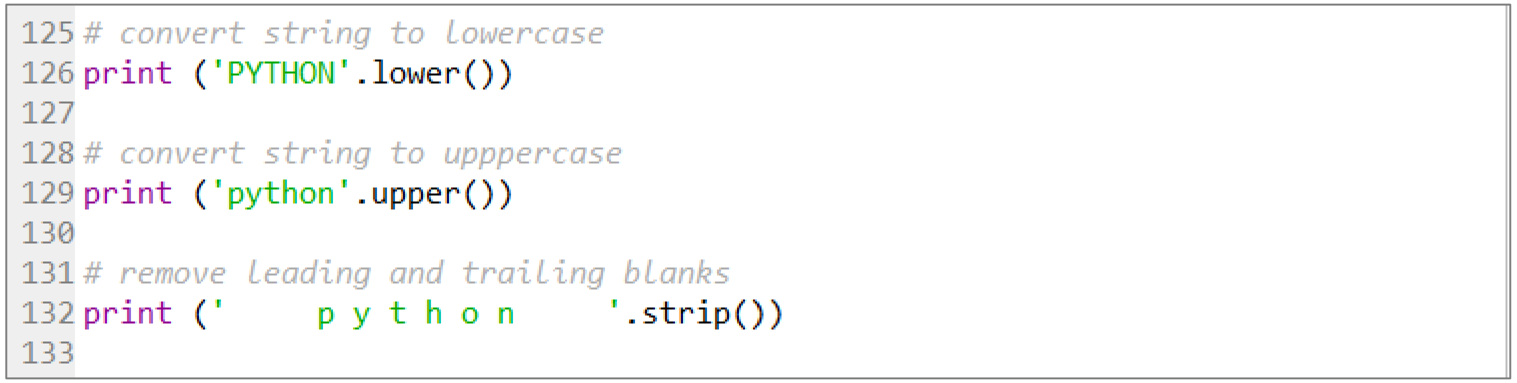
```python
# omitted startIndx is defaulted to 0, omitted endIndx is defaulted to strlen
# this ensures anyString[:n] + anyString[n:] = anyString

print (word[:2])

print (word[4:])

print (word[-2:])

print (word[:4] + word[4:])

print ()

```

```python
# Python strings cannot be changed – they are immutable
# attempting to assign to an index position will result in an error

word[0] = 'J'

```

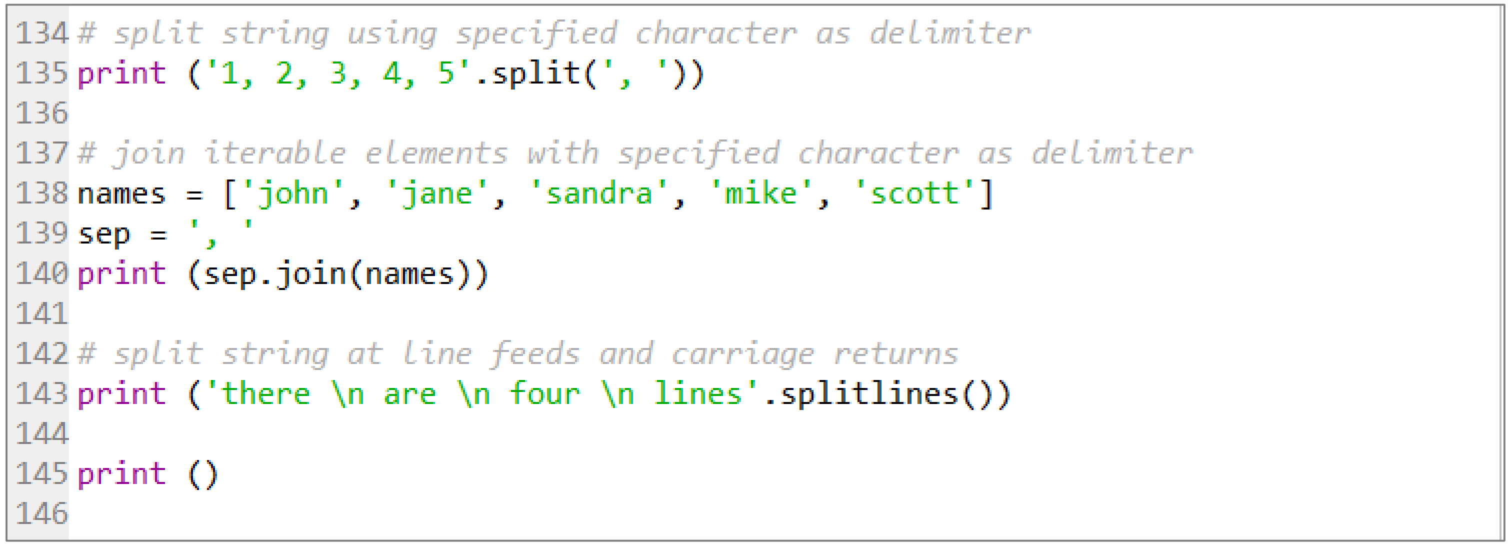
```python
# if you need a different string, you should just create a new one

newword = 'J' + word[1:]

print (newword)

print ()

```



```python
# the Python standard library provides many different methods to  manipulate srtings
# https://docs.python.org/3/library/stdtypes.html#string-methods
# below are ones that are most frequently used

# length of string
print (len('Python'))

# check if substring is in string
print ('th' in 'Python')

# find index of substring in string
print ('Python'.find('th'))

# check if string start with substring
print ('Python'.startswith('Py'))

# check if string ends with substring
print ('Python'.endswith('Py'))

```

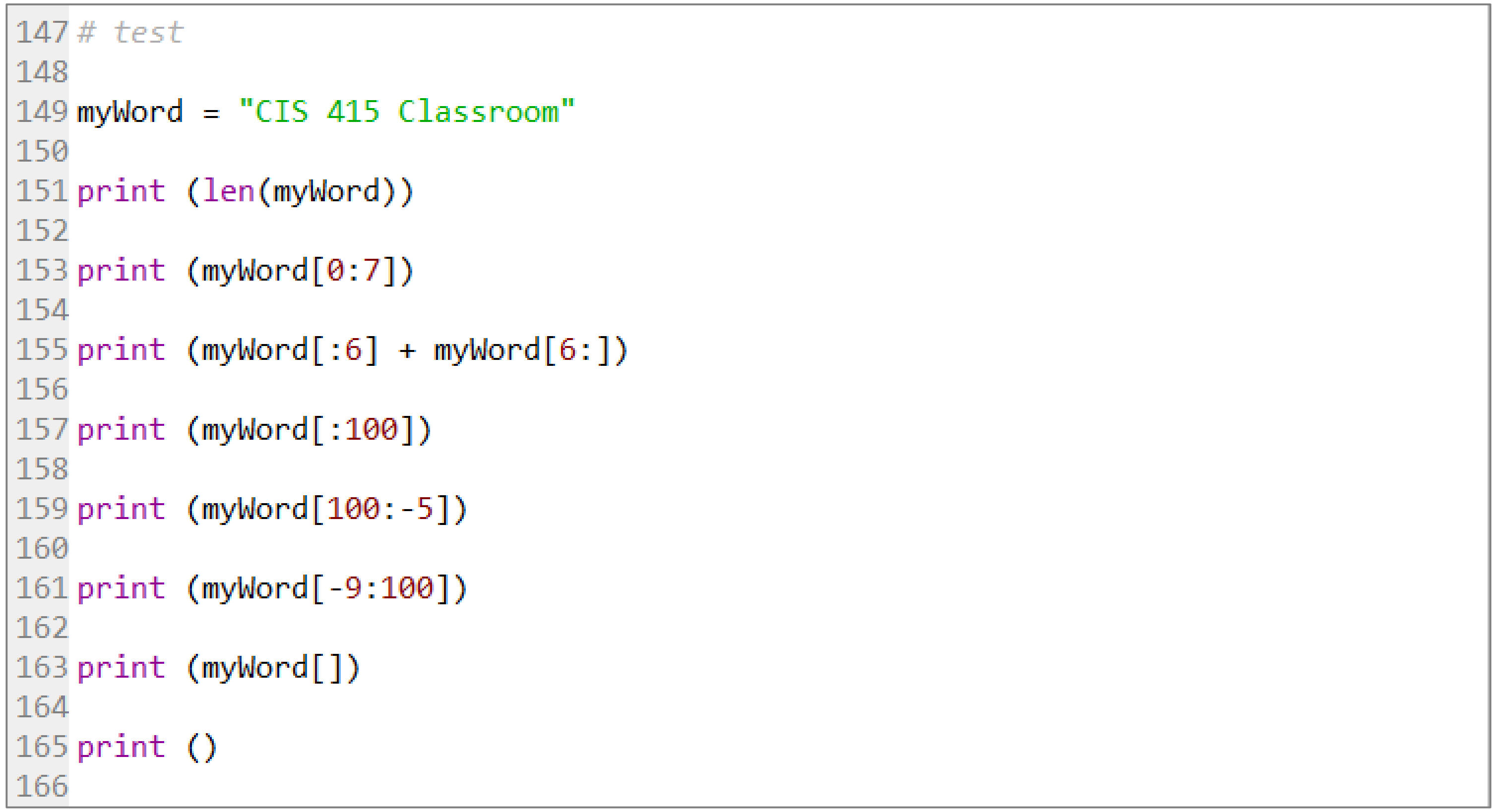


```python
# convert string to lowercase
print ('PYTHON'.lower())

# convert string to upppercase
print ('python'.upper())

# remove leading and trailing blanks
print ('     p y t h o n     '.strip())

```



```python
# split string using specified character as delimiter
print ('1, 2, 3, 4, 5'.split(', '))

# join iterable elements with specified character as delimiter
names = ['john', 'jane', 'sandra', 'mike', 'scott']
sep = ', '
print (sep.join(names))

# split string at line feeds and carriage returns
print ('there \n are \n four \n lines'.splitlines())

print ()

```

Will these statements work?



```
# test

myWord = "CIS 415 Classroom"

print (len(myWord))

print (myWord[0:7])

print (myWord[:6] + myWord[6:])

print (myWord[:100])

print (myWord[100:-5])

print (myWord[-9:100])

print (myWord[])

print ()

```

Lists

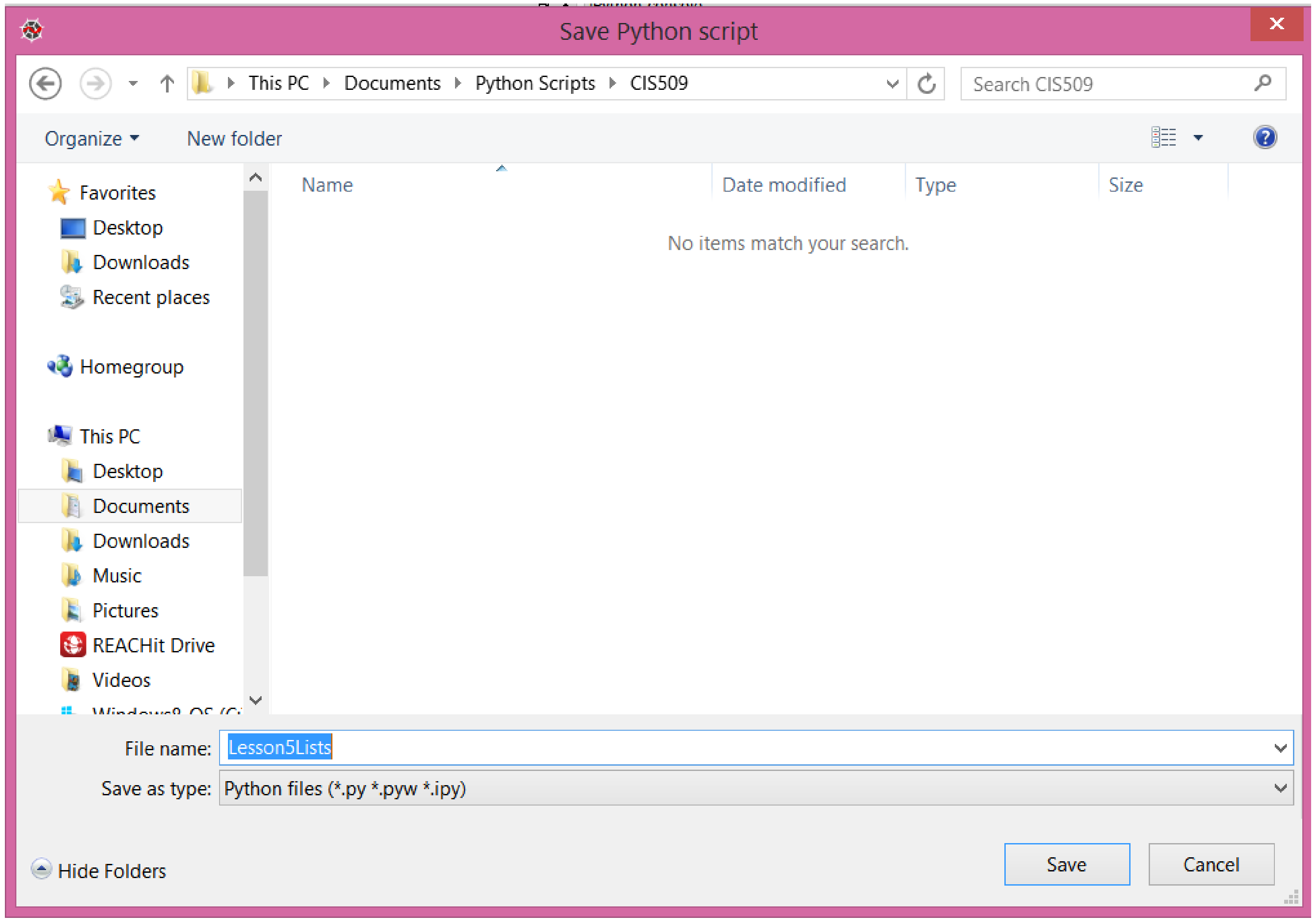
follow along!

(Reference: https://docs.python.org/3.1/tutorial/datastructures.html)

Lists are mutable sequences typically used to store collections of homogeneous items. Lists are indexed by number.
[a, b, c]

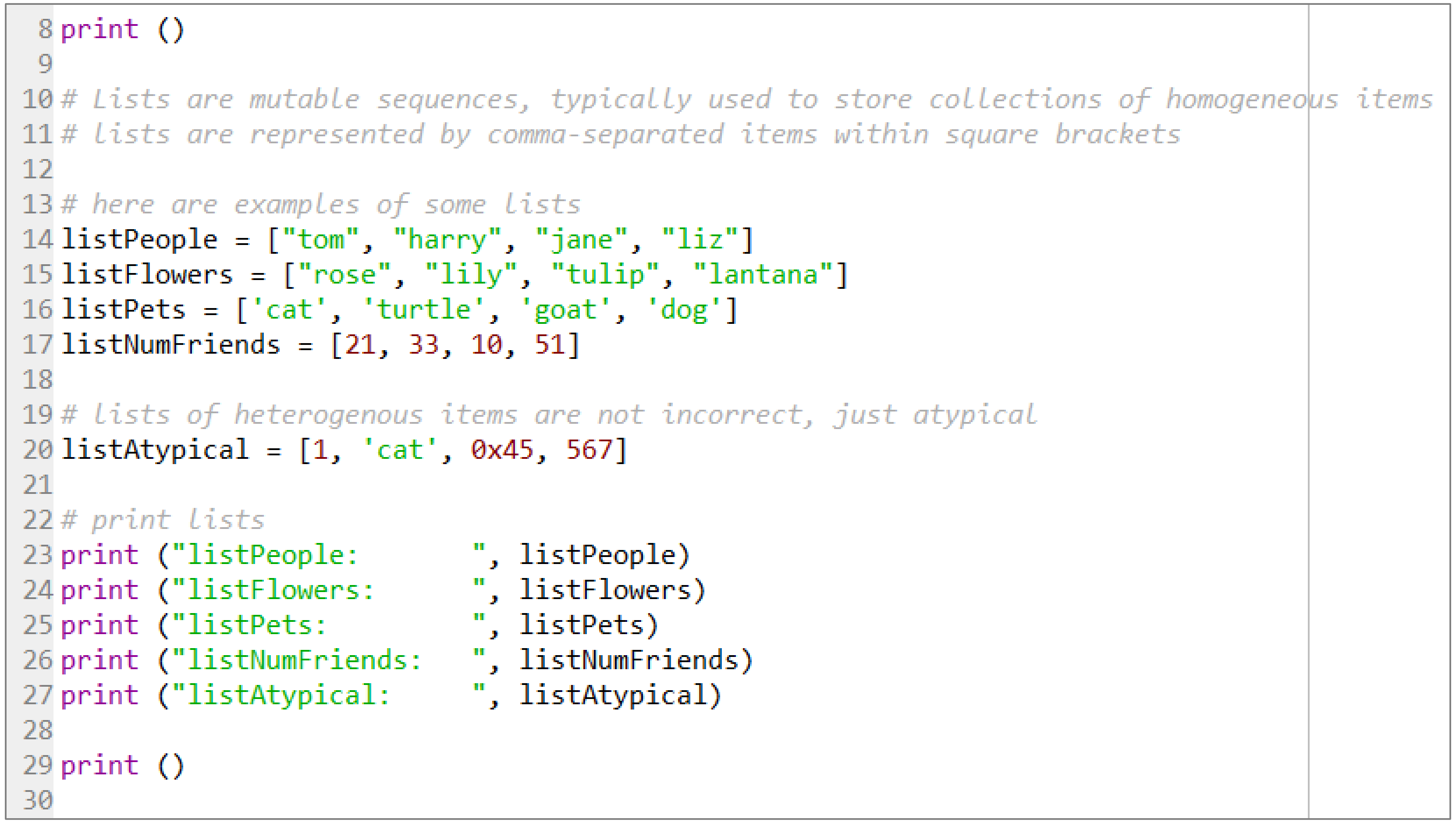
Create a new file called “Lesson5Lists.py”

- Go to File -> New file...
- Go to File -> Save as...
- Go to CISPy directory
- Save file as “Lesson5Lists.py”

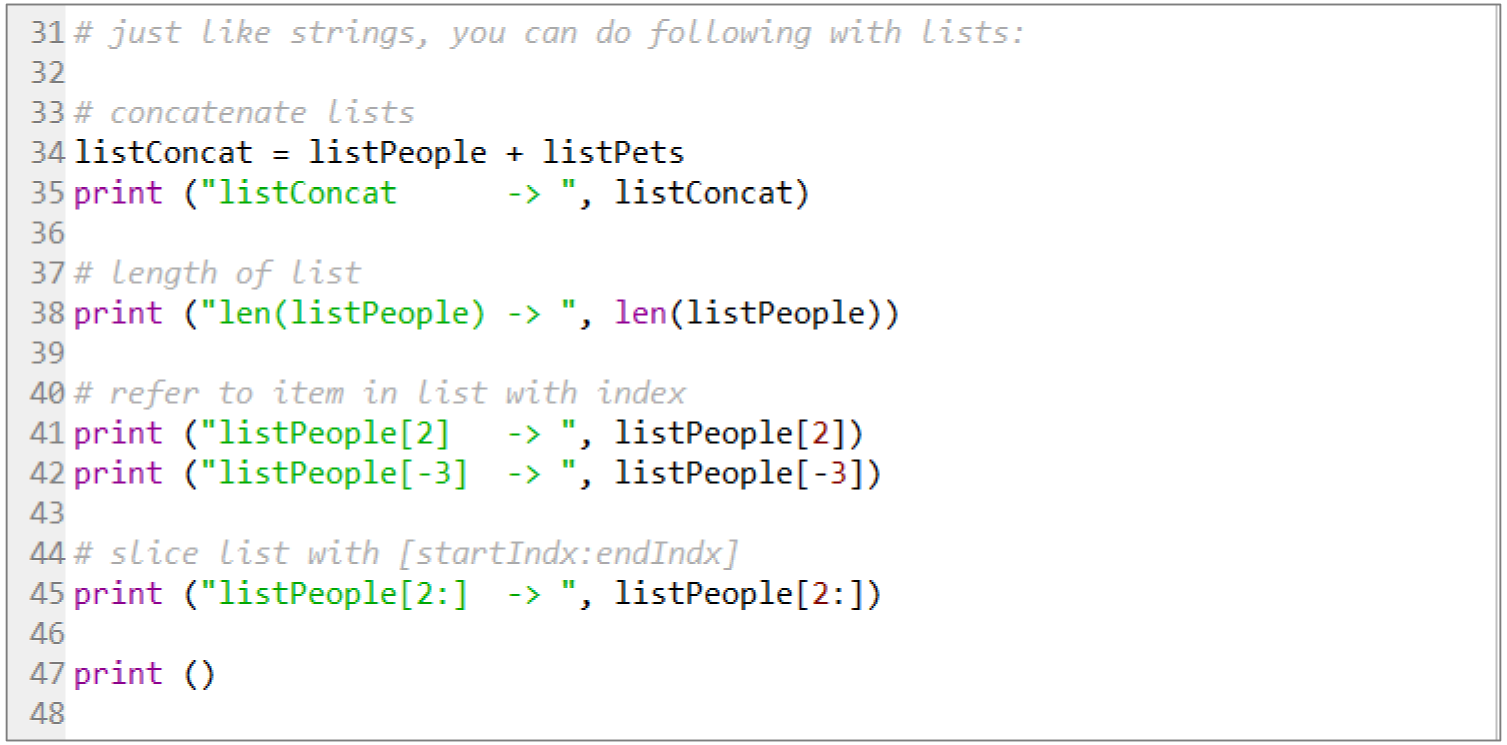


No items match your search.

File name: Lesson5Lists
Save as type: Python files (*.py *.pyw *.ipy)

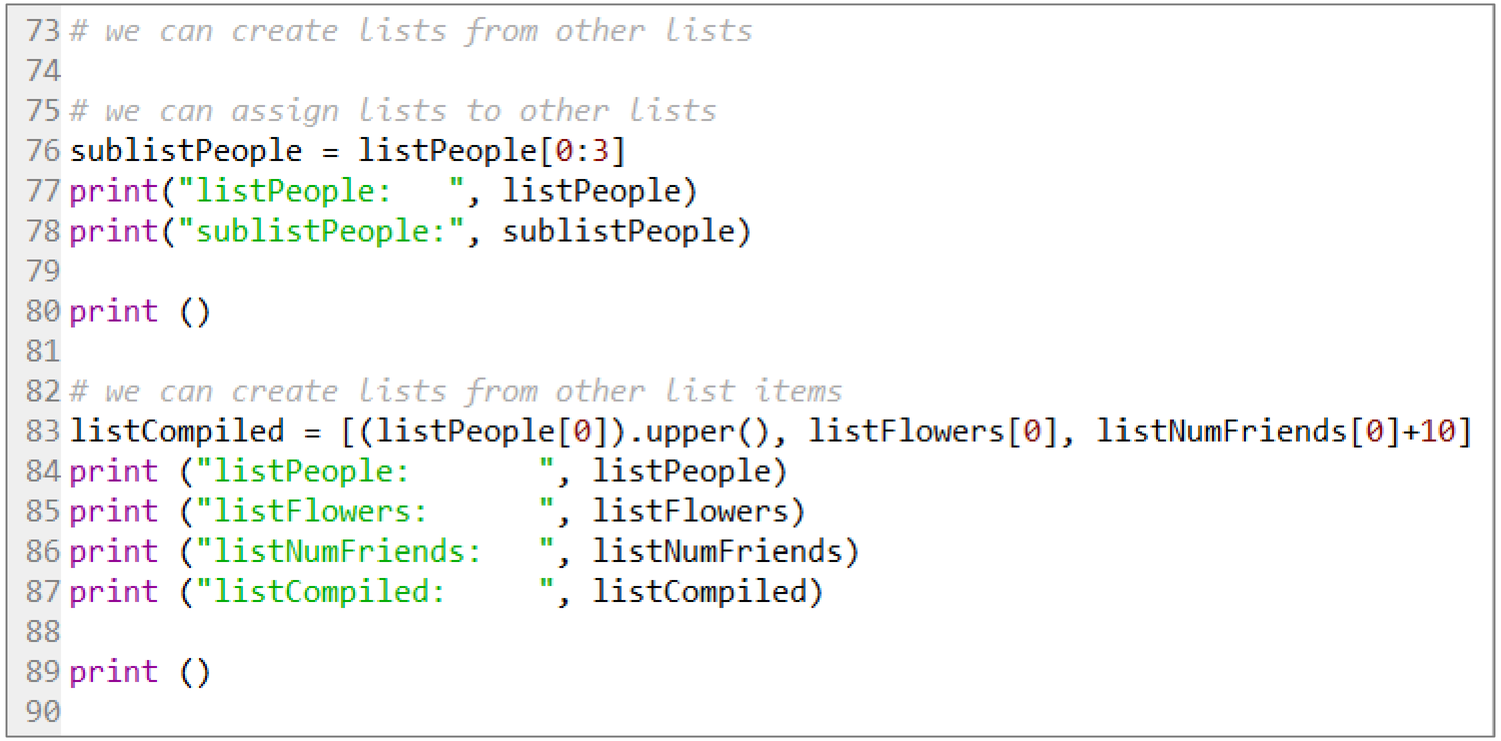


```python
print ()

# Lists are mutable sequences, typically used to store collections of homogeneous items
# lists are represented by comma-separated items within square brackets

# here are examples of some lists
listPeople = ["tom", "harry", "jane", "liz"]
listFlowers = ["rose", "lily", "tulip", "lantana"]
listPets = ['cat', 'turtle', 'goat', 'dog']
listNumFriends = [21, 33, 10, 51]

# lists of heterogenous items are not incorrect, just atypical
listAtypical = [1, 'cat', 0x45, 567]

# print lists
print ("listPeople:       ", listPeople)
print ("listFlowers:      ", listFlowers)
print ("listPets:         ", listPets)
print ("listNumFriends:   ", listNumFriends)
print ("listAtypical:     ", listAtypical)

print ()

```

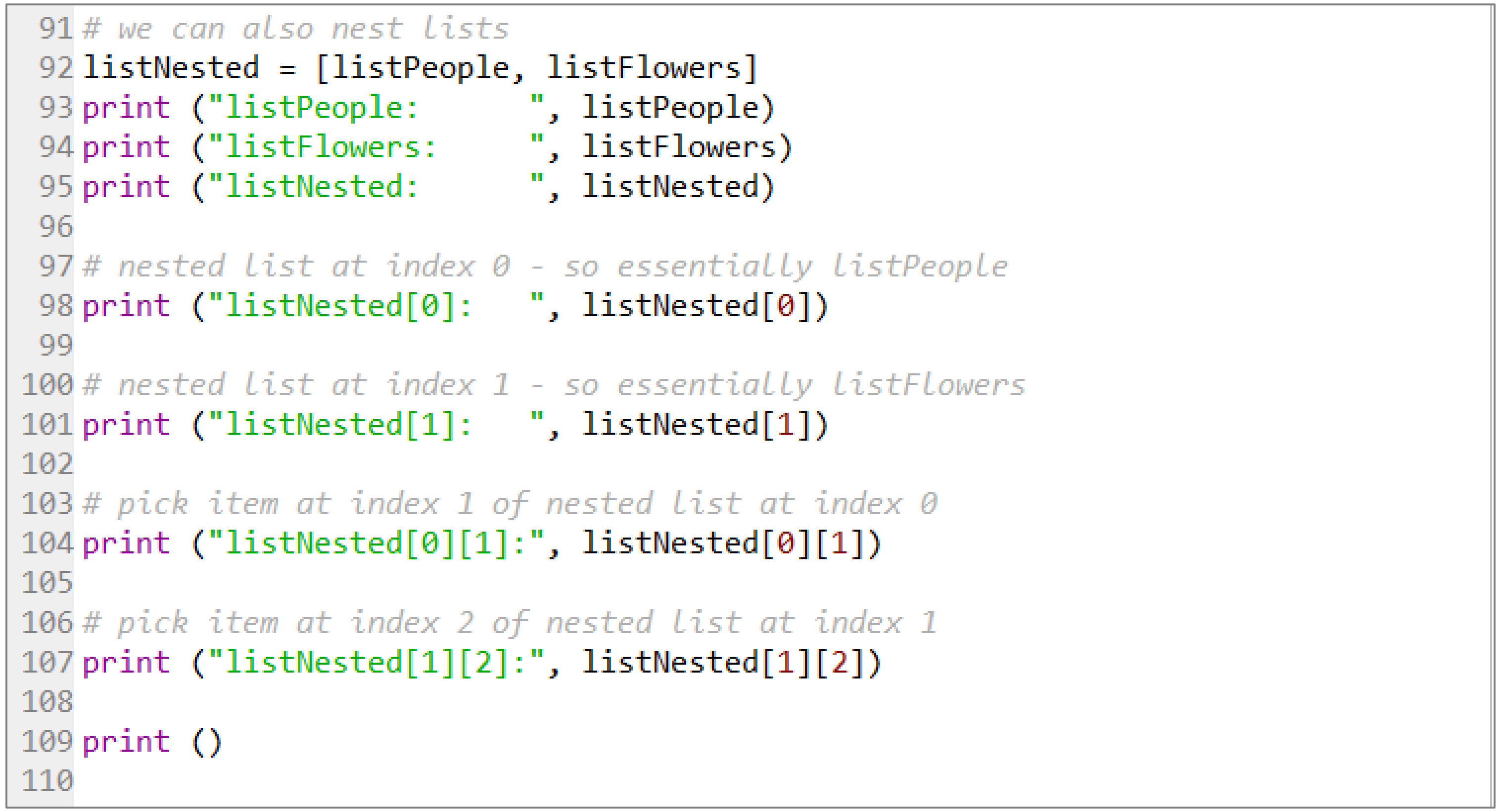


```python
# just like strings, you can do following with lists:

# concatenate lists
listConcat = listPeople + listPets
print ("listConcat      -> ", listConcat)

# length of list
print ("len(listPeople) -> ", len(listPeople))

# refer to item in list with index
print ("listPeople[2]   -> ", listPeople[2])
print ("listPeople[-3]  -> ", listPeople[-3])

# slice list with [startIndx:endIndx]
print ("listPeople[2:]  -> ", listPeople[2:])

print ()

```

```python
# unlike strings, lists are mutable:

# assign to an index
listPets[0] = 'trex'
print (listPets)

# assign to a slice
listPets[0:2] = ['python', 'elephant']
print (listPets)

# delete a slice
listPets[2:4] = []
print (listPets)

# append new items to a list
listPets.append('fox')
print (listPets)

# clear a list by assigning to an empty list
listPets[:] = []
print (listPets)

print ()

```



```python
# we can create lists from other lists

# we can assign lists to other lists
sublistPeople = listPeople[0:3]
print("listPeople:   ", listPeople)
print("sublistPeople:", sublistPeople)

print ()

# we can create lists from other list items
listCompiled = [(listPeople[0]).upper(), listFlowers[0], listNumFriends[0]+10]
print ("listPeople:       ", listPeople)
print ("listFlowers:      ", listFlowers)
print ("listNumFriends:   ", listNumFriends)
print ("listCompiled:     ", listCompiled)

print ()

```

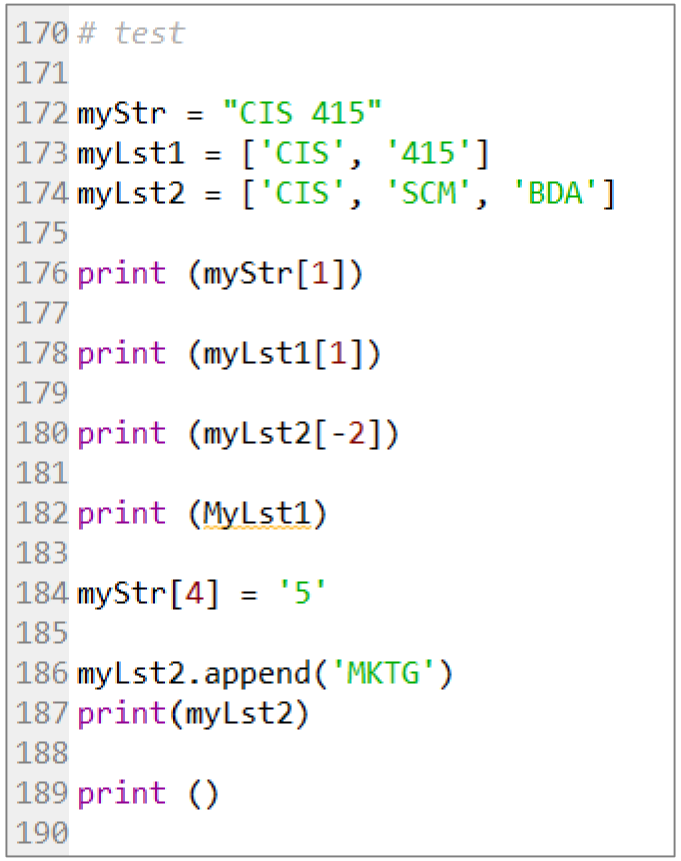


```python
# we can also nest lists
listNested = [listPeople, listFlowers]
print ("listPeople:      ", listPeople)
print ("listFlowers:     ", listFlowers)
print ("listNested:      ", listNested)

# nested list at index 0 - so essentially listPeople
print ("listNested[0]:   ", listNested[0])

# nested list at index 1 - so essentially listFlowers
print ("listNested[1]:   ", listNested[1])

# pick item at index 1 of nested list at index 0
print ("listNested[0][1]:", listNested[0][1])

# pick item at index 2 of nested list at index 1
print ("listNested[1][2]:", listNested[1][2])

print ()

```

```python
# the Python standard library provides many different methods to  manipulate lists
# https://docs.python.org/3/tutorial/datastructures.html
# below are ones that are most frequently used

lst1 = [100, 200, 300]
lst2 = [5, 15, 25]
lst3 = [10, 50, 50, 20, 0, 10, 50]
print ("lst1: ", lst1)
print ("lst2: ", lst2)
print ("lst3: ", lst3)

# Add an item to the end of the list. Equivalent to lst1[len(lst1):] = [-6000]
lst1.append(-400)
print ("lst1: ", lst1)

# Extend the lst2 by appending all the items in lst1. Equivalent to lst2[len(lst2):] = lst1
lst2.extend(lst1)
print ("lst2: ", lst2)

# Insert an item at index [3]
lst2.insert (3, -400)
print ("lst2: ", lst2)

# Remove the first item from lst2 whose value is -400.
lst2.remove(-400)
print ("lst2: ", lst2)

# Remove the item at index [6]
del lst2[6]
print ("lst2: ", lst2)

# Remove and returns the last item in the list.
lst2.pop()
print ("lst2: ", lst2)

```

```python
# Return the index in lst3 of the first item whose value is 50
print (lst3.index(50))

# Return the number of times 50 appears in lst3
print (lst3.count(50))

# Reverse the items of lst3 in place.
lst3.reverse()
print ("lst3: ", lst3)

# Sort the items of lst3 in place.
lst3.sort()
print ("lst3: ", lst3)

# Remove all items from the list
lst3.clear()
print ("lst3: ", lst3)

# Delete the list - so now any refrence to the list will be an error
del lst3
#print ("lst3: ", lst3)

print ()

```

Will these statements work?



```
# test

myStr = "CIS 415"
myLst1 = ['CIS', '415']
myLst2 = ['CIS', 'SCM', 'BDA']

print (myStr[1])

print (myLst1[1])

print (myLst2[-2])

print (MyLst1)

myStr[4] = '5'

myLst2.append('MKTG')
print(myLst2)

print ()

```

Tuples

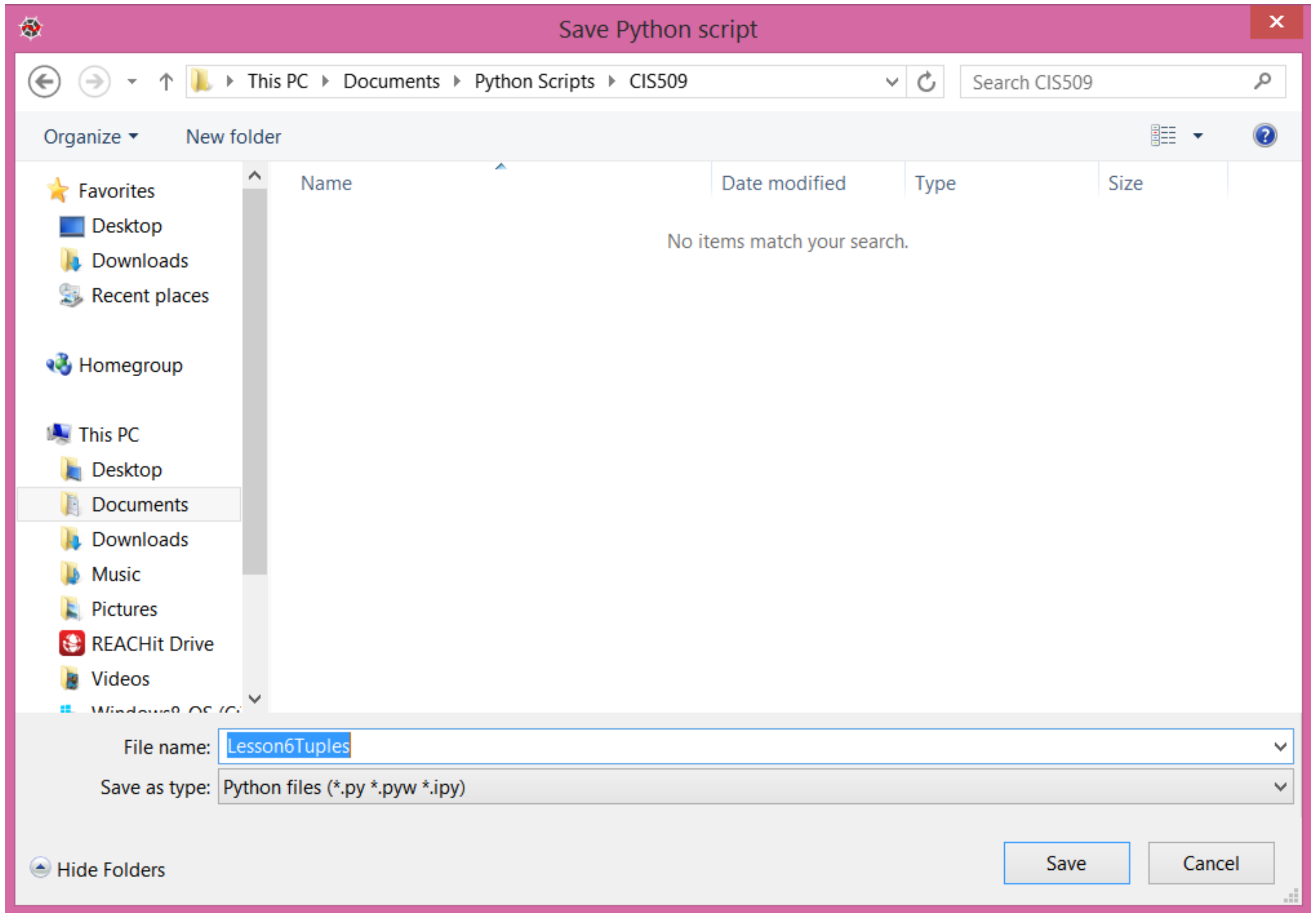
follow along!

(Reference: https://docs.python.org/3.1/tutorial/datastructures.html)

Tuples are immutable sequences typically used to store collections of heterogeneous data . Tuplesare indexed by number.
(a, b, c)

Create a new file called “Lesson6Tuples.py”

- Go to File -> New file…
- Go to File -> Save as…
- Go to CISPy directory
- Save file as “Lesson6Tuples.py”



No items match your search.

File name: Lesson6Tuples

Save as type: Python files (*.py *.pyw *.ipy)

```python
# declaring empty tuples

tuple1 = ()
print ("tuple1: ", tuple1, len(tuple1))

tuple2 = tuple ()
print ("tuple2: ", tuple2, len(tuple2))

print ()

```

```python
# singleton tuples must have a trailing comma

tuple3 = (1,) # tuple3 = (1) will throw a syntax error
print ("tuple3: ", tuple3, len(tuple3))

print ()

```

```python
# tuple operations: limited since unlike lists, tuples are immutable

tuple4 = (1, 2, 3)

# you can get length of tuples just like lists
print ("tuple4: ", tuple4, len(tuple3))

# you can index tuples just like lists
print ("tuple4: ", tuple4[0], tuple4[1], tuple4[2])

# you can slice tuples just like lists
print ("tuple4[0:2]: ", tuple4[0:2])

print ()

```

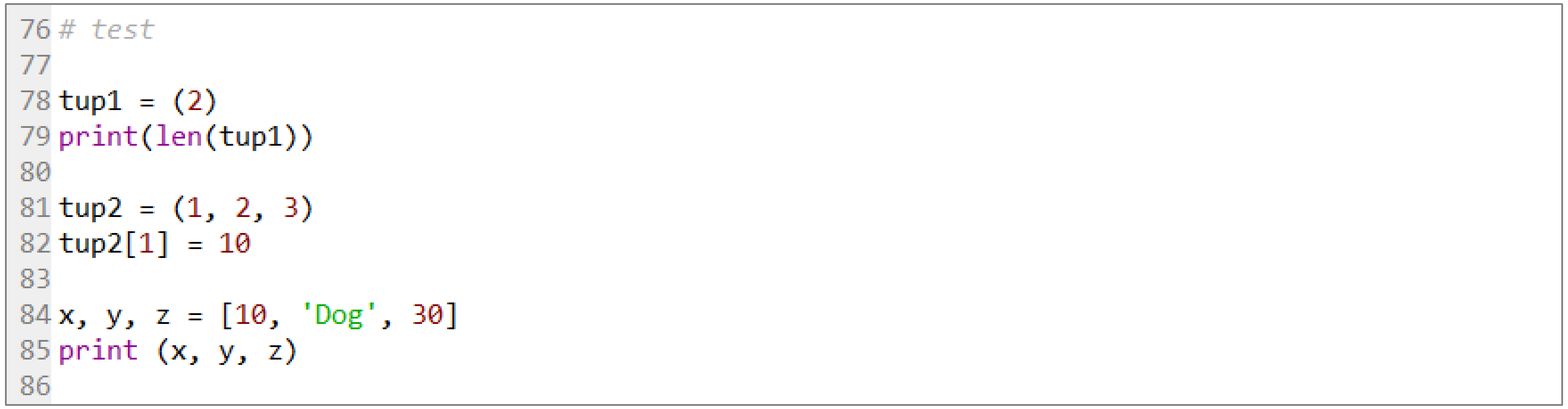
```python
# you can also nest tuples like lists
tuple5 = ((1, 2, 3), (4, 5, 6))
print ("tuple5: ", tuple5, len(tuple5))
print ("tuple5[1][2]: ", tuple5[1][2])

print ()

```

```python
# you can convert lists to tuples
tuple6 = tuple ([1, 2, 3])
print ("tuple6: ", tuple6, len(tuple6))

print ()

```

```python
# tuple packing and unpacking
# multiple assignment is essentially packing/unpacking in action

# tuple packing
tuple7 = 1, 2, 3
print ("tuple7: ", tuple7, len(tuple7))

# tuple unpacking
x, y, z = tuple7
print ("unpacked tuple7: ", x, y, z)

```

```python
# Note that it is actually the comma which makes a tuple, not the parentheses.
# The parentheses are optional, except in the empty tuple case, or when they
# are needed to avoid syntactic ambiguity.
# For example, f(a, b, c) is a function call with three arguments,
# while f((a, b, c)) is a function call with a 3-tuple as the sole argument.

```

Will these statements work?



```python
# test

tup1 = (2)
print(len(tup1))

tup2 = (1, 2, 3)
tup2[1] = 10

x, y, z = [10, 'Dog', 30]
print (x, y, z)

```