Programming in Python

Lecture 3

Lists & tuples.

Lists & tuples.

Python data types

Python has 4 types categories:

- **Numbers** (integer, real, complex). Supports addition, multiplication, etc.
- **Sequencies** (string, list, tuple). Support indexing, slice extraction, concatenation, etc..
- **Mappings** (Dictionaries). Support key indexing operation, etc.
- **Sets.** They form a separate category of types (they do not map keys to values and are not ordered sequences).

1. Lists

- **Positionally ordered collections of objects of various types**

They provide a very flexible tool for representing different collections—lists of files in a folder, employees in a company, emails in your inbox, and so on.

- **Access to elements by index (offset)**

You can use the indexing operation to extract individual objects from the list by their offsets, extract slices and do concatenation.

- **Variable-length, heterogeneous, and arbitrarily nestable**

Unlike strings, lists can increase and decrease directly (their length can vary) and can contain any (including complex) objects (lists are heterogeneous). Lists support the ability to create an arbitrary number of nesting levels.

- **Mutable**

Lists can be modified in place by assignment to offsets as well as a variety of list method calls.

- **Arrays of object references**

Formally, lists in Python can contain zero or more pointers to other objects. Lists are somewhat reminiscent of arrays of pointers (addresses). Retrieving an element from a list in Python is as fast as retrieving an element of an array in C.

1.1. Literals & operations

Operation	Interpretation
L = [ ]; L = [0, 1, 2, 3]	Empty list; List from 4 elements
L = [‘abc’, [‘def’, ghi’]]	Nested Lists
L = list(‘spam’)	Creating a list from an iterable object (this way you can copy lists)
L = list(range(-4, 4))	Creation a list from a continuous sequence of integers
L[i]; L[i][j]; L[i:j]	Index; Index of index; Slice
len(L)	Length
L1 + L2; L* 3	Concatenate, repeat
for x in L: print(x)	Iteration
3 in L	Membership
L.append(4), L.extend([5,6,7])	Append 1 or more (extend) elements to the end of the list
L.insert(i, x)	Insert element x at position i
L.index(x)	Find the index of the first occurrence of x
L.count(x)	Count the number of occurrences of an element x
L.sort(); L.reverse()	direct sort; reverse sort
del L[k]; del L[i:j]	Removing list elements
L.pop(i); L.pop()	Removing the specified item from the list; last one. Returns the deleting item.
L.remove(x)	Removing the 1st occurrence of the given element x
';'.join(list)	Combines list items into one line

1.2. Out of range

If you address to a non-existent item, instead of increasing the size of the list, the Python interpreter will throw an error.
You need to **increase the list explicitly** - using the commands **append, extend, insert**.

```
>>> L
[123, ‘spam’, ‘NI’]
>>> L[99]
...  error message text is omitted ...
IndexError: list index out of range

>>> L[99] = 1
...  error message text is omitted ...
IndexError: list assignment index out of range
```

1.3. Examples

```
>>> L = [123, ‘spam’, 1.23] # List of 3 objects from different types
>>> L.append(‘NI’) # Append new element to the end of the list
>>> L
[123, ‘spam’, 1.23, ‘NI’]

>>> L.pop(2) # Remove the specified element from the list
1.23
>>> L # del L[2]  do the same thing
[123, ‘spam’, ‘NI’]

>>> M = [5, 0, 124, 37]
 >>> M.sort() # direct sort

>>> M
[0, 5, 37, 124]

>>> M.reverse() # reverse sort
>>> M
[124, 37, 5, 0]
```

1.3. Examples

```
# This is my shopping list
shoplist = [‘apples’, ‘mango’, ‘carrot’, ‘bananas']
print(‘I have to make ', len(shoplist), ' purchases.')
print(‘Purchases :', end=' ')
for item in shoplist:  print(item, end=' ')

print(‘\n Also, need to buy rice.') shoplist.append(‘rice')
print(Now my shopping list is:', shoplist)

print(“I'll sort my list “)
shoplist.sort()
print(This is sorted list:', shoplist)

print(' The first thing I need to buy is ', shoplist[0])
olditem = shoplist[0]
del shoplist[0]
print(I’ve bought', olditem)
print(‘Now my shopping list is :', shoplist)
```

Program print:

```
I have to make 4 purchases.
Purchases: apples mango carrot banana.

Also, need to buy rice.
Now my shopping list is:
 [‘apples’, ‘mango’, ‘carrot’, ‘bananas ’, ’rice’]
I'll sort my list

This is sorted list :
[‘apples’, bananas ’, ‘carrot’, ‘mango’, ‘ rice’]
The first thing I need to buy is apples


I’ve bought apples
Now my shopping list is:
 [‘bananas ’, ‘carrot’, ‘mango’, ‘ rice’]
```

1.3. Examples

```
# Get the digits  and
# put them on the list digits

s = input(‘Input the string: ')
digits = []
for symbol in s:
    if '1234567890'.find(symbol) != -1:
        digits.append(int(symbol))
print(digits)
```

1.4. Split & Join example

```
# string to list
str = "this is a string"
lst = str.split(" ") # str is converted to a list of strings.
print (lst)
['this', 'is', 'a', 'string’]

# list to string
str = "-".join(lst)
print (str)
this-is-a-string
```

1.5. List generator

A list generator is a way to build a new list by applying an expression to each element of a sequence.

```
# Create a list from  triple letters of the string ’list’
>>> c = [c * 3 for c in 'list']
>>> c
['lll', 'iii', 'sss', 'ttt']

# Create a list from  triple letters of the string ’list’, except  the letter ’i’
>>> c = [c * 3 for c in 'list' if c != 'i']
>>> c
['lll', 'sss', 'ttt']

# Nested loop
>>> c = [c + d for c in 'list' if c != 'i' for d in 'spam' if d != 'a']
>>> c
['ls', 'lp', 'lm', 'ss', 'sp', 'sm', 'ts', 'tp', 'tm']
```

1.5. List generator

```
# Nested loop
>>> c = [c + d for c in 'list' if c != 'i' for d in 'spam' if d != 'a']
>>> c
['ls', 'lp', 'lm', 'ss', 'sp', 'sm', 'ts', 'tp', 'tm']
```

```
res0=[]
for s1 in 'list':
    for s2 in 'spam':
        if s1 != 'i' and s2 != 'a':
            res0.append(s1+s2)
print('res0=', res0)
```

(a)

```
res=[]
for s1 in 'list':
    if s1 != 'i':
        for s2 in 'spam':
            if s2 != 'a':
                res.append(s1+s2)
print('res=',res)
```

(б)

1.6. Function map

map(func, seq)

func — function name, ***seq*** — sequence.

map applies the function ***func*** to all elements of sequence **seq**.

The map object is an iterator over our results, so we could loop over it with **for** or we can use **list()** to turn it into a list.

list (**map(func, seq) —** present the result as a list.

```
# Find the number of positive elements in
# this list
a = list(map(int, input().split()))
count = 0
for i in range(len(a)):
    if a[i] > 0:
        count += 1
print(count)
```

2. Pointers and mutable objects

```
>>> L1 = [2, 3, 4]  # mutable object
>>> L2 = L1 # second pointer at the same object
>>> L1[0] = 24  # changing object

>>> L1  # The list L1 changed
[24, 3, 4]
>>> L2 # and the list L2 changed as well !
[24, 3, 4]
```

```
>>> L1 = [2, 3, 4]  # mutable object
>>> L2 = L1[:]  # creating a copy of the list L1
>>> L1[0] = 24  # changing object

>>> L1
[24, 3, 4]
>>> L2  # L2 didn’t change
[2, 3, 4]
```

3. Pointers and equality

```
>>> L = [1, 2, 3]
>>> M = L # M and L  point to the same object
>>> L == M # Same value
True
>>> L is M # Same object
True
```

```
>>> L = [1, 2, 3]
>>> M = [1, 2, 3] # M and L  point to the different objects
>>> L == M  # Same value
True
>>> L is M # But different objects
False
```

```
>>> X = 42
>>> Y = 42  # It should be two different objects
>>> X == Y
True
>>> X is Y  # But X and Y actually point to the same object (because of cache)!
True
```

4. Tuples

- **Ordered collections of arbitrary objects**

Like strings and lists, tuples are positionally ordered collections of objects (i.e., they maintain a left-to-right order among their contents); like lists, they can embed any kind of object.

- **Accessed by offset**

Like strings and lists, items in a tuple are accessed by offset (not by key); they support all the offset-based access operations, such as indexing and slicing.

- **Of the category “immutable sequence”**

Like strings and lists, tuples are sequences; they support many of the same operations. However, like strings, tuples are immutable; they don’t support any of the in-place change operations applied to lists.

- **Fixed-length, heterogeneous, and arbitrarily nestable**

Because tuples are immutable, you cannot change the size of a tuple without making a copy. On the other hand, tuples can hold any type of object, including other compound objects (e.g., lists, dictionaries, other tuples), and so support arbitrary nesting.

- **Arrays of object references**

Like lists, tuples are best thought of as object reference arrays; tuples store access points to other objects (references), and indexing a tuple is relatively quick.

4.1. Literals & operations

Operation	Interpretation
T = ( ); T = (0,)	An empty tuple; A one-item tuple (comma !)
T = (0, ‘Ni’, 1.2, 3); T = 0, ‘Ni’, 1.2, 3	A four-item tuple
T = (‘abc’, (‘def’,‘ghi’))	Nested tuples
T = tuple(‘spam’)	Tuple of items in an iterable
T[i]; T[i][j]; T[i:j]	Index, index of index, slice
len(T)	length
T1 + T2	Concatenate
T* 3	Repeat
for x in T: print(x)	Iteration
‘spam’ in T2	Membership
T.index(x)	Find the index of the first occurrence of x
T.count(x)	Count the number of occurrences of an element x
sorted(T)	Sorting

4.2. Examples

```
>>> (1, 2) + (3, 4) # concatenate
(1, 2, 3, 4)

>>> (1, 2) * 4 # repeat
(1, 2, 1, 2, 1, 2, 1, 2)

# Index, slice
>>> T = (1, 2, 3, 4)
>>> T[0], T[1:3]
(1, (2, 3))

# Swap variable values
a, b = b, a
```

```
# Let’s create a one-item tuple
>>> a = ('s’)
>>> a
 's’
Got a string  :-(
# What is the right way?
>>> a = ('s', )
>>> a
('s',)


# Tuple of items in an iterable
>>> a = tuple('hello, world!')
>>> a
('h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')
```

4.2. Examples

```
zoo = (‘python', ‘elephant', ‘pinguin') # parentheses are optional
print('The number of animals in the zoo-', len(zoo))
new_zoo = ‘monkey’, ‘camel', zoo
print(‘Number of cages in the zoo -', len(new_zoo))
print('All animals in the new zoo:', new_zoo)

print('Animals brought from the old zoo:', new_zoo[2])
print('Last animal brought from the old zoo -', new_zoo[2][2])
print(' The number of animals in the new zoo -', len(new_zoo)-1+len(new_zoo[2]))
```

```
The number of animals in the zoo- 3

Number of cages in the zoo - 3
All animals in the new zoo :
('monkey’, camel’, (python', 'elephant', ‘pinguin '))
Animals brought from the old zoo: ('python', ‘elephant', 'pinguin')
 Last animal brought from the old zoo - pinguin
The number of animals in the new zoo - 5
```

4.3. Function zip

Function zip takes one or more sequences as arguments and returns a series of tuples that pair up parallel items taken from those sequences.

zip stops execution as soon as the end of the shortest list is reached.

```
a = [1,2,3]
b = "xyz"
c = (None, True)

res = list(zip(a, b, c))
print (res)

[(1, 'x', None), (2, 'y', True)]
```

4.4. Usage

Why have a type that is like a list, but supports fewer operations? Frankly, tuples are not generally used as often as lists in practice, but their immutability is the whole point.

- **Tuples provide a sort of integrity constraint that is convenient in large programs**

If you pass a collection of objects around your program as a list, it can be changed anywhere; if you use a tuple, it cannot.

- **Takes up less memory than a similar list**

```
>>> a = (1, 2, 3, 4, 5, 6)
>>> b = [1, 2, 3, 4, 5, 6]
>>> a.__sizeof__()
72
>>> b.__sizeof__()
88
```

- **It is possible to use as dictionary keys**
- **Mass initialization of variables:** (a, b, c) = (1, 2.08, 'Yes’).
- **Return multiple values from a function at the same time.**
- For example, we want to calculate the moves of a horse in chess:

```
dx = (1, 2, 2, 1, -1, -2, -2, -1)
dy = (2, 1, -1, -2, -2, -1, 1, 2)
for i in range(8): potential move: x+dx[i], y+dy[i]
```

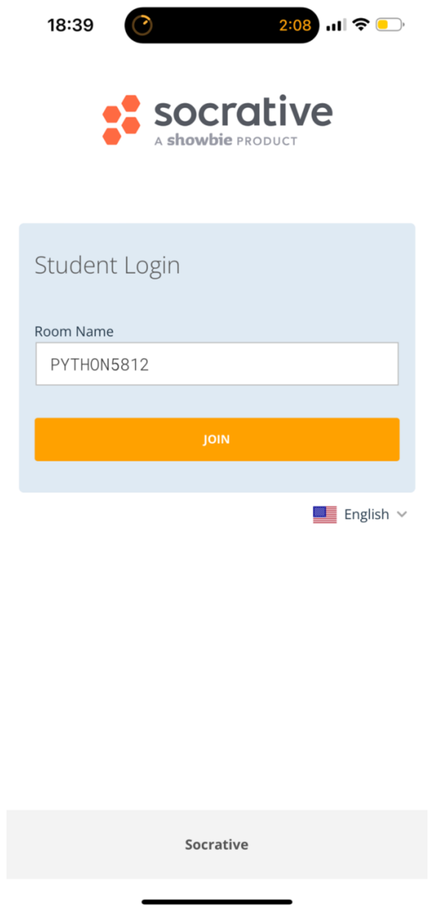
5. Unpacking sequences

This is a feature of the Python language that allows you to specify a variable that will have a list of values and unpack the sequence into it.

```
a, *b, c = range(10) #a = 0, b = [1, 2, 3, 4, 5, 6, 7, 8], c = 9
d, *e = range(10) #d = 0, e = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Quiz with Socrative

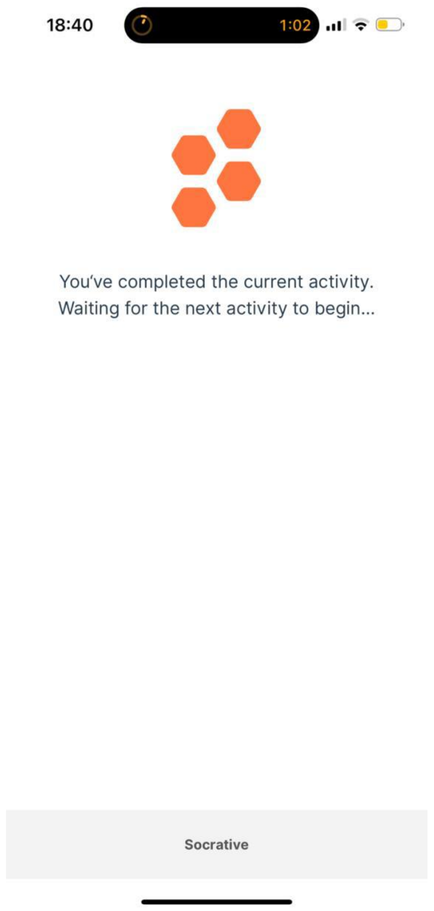
1. Install mobile application **Socrative Student** or go to the https://www.socrative.com/
2. Join Room name **PYTHON5812**
3. Put your Name in the format: ***Surname_name_subgroup number***



Student Login

Room Name

PYTHON5812



You've completed the current activity.
Waiting for the next activity to begin...

Contest 2

https://official.contest.yandex.ru/contest/67988/enter/