<https://zoom.us/j/541492863>

**Question:1**

numbers = [ 951, 402, 984, 651, 360, 69, 408, 319, 601, 485, 980, 507, 725, 547, 544, 615, 83, 165, 141, 501, 263, 617, 865, 575, 219, 390, 984, 592, 236, 105, 942, 941, 386, 462, 47, 418, 907, 344, 236, 375, 823, 566, 597, 978, 328, 615, 953, 345, 399, 162, 758, 219, 918, 237, 412, 566, 826, 248, 866, 950, 626, 949, 687, 217, 815, 67, 104, 58, 512, 24, 892, 894, 767, 553, 81, 379, 843, 831, 445, 742, 717, 958, 609, 842, 451, 688, 753, 854, 685, 93, 857, 440, 380, 126, 721, 328, 753, 470, 743, 527]

Copy and Paste the above list in your code to create list with name numbers.

Loop through and print out all even numbers from the numbers list in the same order they are received. Don't print any numbers that come after 237 in the sequence.

**Question:2**

Solve the Below String Related Assignments

1.

#Given Input-1 :-SAKEEB

#Given Input-2 :-SHEIKH

#Expected Output :- SSAHKEEIEKBH

SS AH KE EI EK BH

**SS AH KE EI EK BH**

**SS AH KE EI EK BH**

**SAKEEBSHEIKH**

**SS HEIKHASKHEEEIBKSH**

**NOTE :- You have to submit the assignment on Monday (04-Mar-19)**

Presentation:

1. Jeremy --> Fundamental DataTypes

2. Michael --> Features of List, Tuple, Dictionary

3. Ann --> Operators in python

4. Gary --> conditional If Statement

5. Stan --> Loop Structures

6. Zachary --> Print Methods

Question:1

numbers = [ 951, 402, 984, 651, 360, 69, 408, 319, 601, 485, 980, 507, 725, 547, 544, 615, 83,

165, 141, 501, 263, 617, 865, 575, 219, 390, 984, 592, 236, 105, 942, 941, 386, 462, 47, 418,

907, 344, 236, 375, 823, 566, 597, 978, 328, 615, 953, 345, 399, 162, 758, 219, 918, 237, 412,

566, 826, 248, 866, 950, 626, 949, 687, 217, 815, 67, 104, 58, 512, 24, 892, 894, 767, 553, 81,

379, 843, 831, 445, 742, 717, 958, 609, 842, 451, 688, 753, 854, 685, 93, 857, 440, 380, 126,

721, 328, 753, 470, 743, 527]

Copy and Paste the above list in your code to create list with name numbers.

Loop through and print out all even numbers from the numbers list in the same order they are

received. Don&#39;t print any numbers that come after 237 in the sequence.

Question:2

Solve the Below String Related Assignments

1.

#Given Input-1 :-SAKEEB

#Given Input-2 :-SHEIKH

#Expected Output :- SSAHKEEIEKBH

SS AH KE EI EK BH

SS AH KE EI EK BH

SS AH KE EI EK BH

SS AH KE EI EK BH

NOTE :- You have to submit the assignment on Monday (04-Mar-19)

Note :- Need to present topics on monday(04/03/19)

Why we need python lists?

Lists and tuples are sequential data

what's the point of using a tuple in the first place?

Immutability?

**Tuples have structure, lists have order.**

One example would be pairs of page and line number to reference locations in a book, e.g.:  
my\_location = (42, 11)  # page number, line number

One common summary of these more interesting, if subtle, differences is that tuples are heterogeneous and lists are homogeneous. In other words:

* Tuples (generally) are sequences of different kinds of stuff, and you deal with the tuple as a coherent unit.
* Lists (generally) are sequences of the same kind of stuff, and you deal with the items individually.
* Consider the following two data structures:
* >>> import time
* >>> time.localtime()
* (2008, 2, 5, 11, 55, 34, 1, 36, 0)
* >>> range(10)
* [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

The first one, a tuple, is a sequence in which position has semantic value. The first position is always a year. This tuple functions as a lightweight record or struct.

The second one, a list, is a sequence where we may care about [order](http://news.e-scribe.com/397), but where the individual values are functionally equivalent.

You would use tuples to represent a point (x,y): that alone already gives away the big semantic difference between tuples and lists and helps clarify that they are not just read-only lists.

**Mutable vs. Immutable Types**

Python built-in types can be split into two main categories — *Mutable* and *Immutable* types. Mutabilityis just a fancy way of specifying if an object can be modified after it has been declared.

Lists and dictionaries are excellent types to use when you know your user’s data is going to grow over time because of their mutability characteristics. As a result they are very often used in real world business applications.

List objects offer many built-in methods (a function that belongs to an object class and its instances)such as append() and pop() which add and remove list elements from the end of a list.

List objects offer many built-in methods (a function that belongs to an object class and its instances)such as append() and pop() which add and remove list elements from the end of a list.

Let’s walk through ordering food at a restaurant in OOP:

You start off with an empty order:

order = []

Let’s inspect some of these attributes real quick just to make sure we have a list:

>>> type(order)  
<class ‘list’>

>>> id(order)  
**140201387356936**

Next, the server comes and we order a few items:

>>> order.append(“steak”, “salad”, “potatoes”)  
>>> order  
[‘steak’, ‘salad’, ‘potatoes’]

OH WAIT! We don’t want regular potatoes, we want sweet potatoes!

>>> order.pop()  
>>> order  
[‘steak’, ‘salad’]  
>>> order.append("sweet potaters")  
>>> order  
[‘steak’, ‘salad’, 'sweet potaters']  
>>> id(order)  
140201387356936



As you can see, we are able to run methods on the list. This does not return a new list, but mutates (changes) the original list. In the end, our two calls to id(cart) return the same integer proving cart to be the same list before and after list method invocations. This is a hallmark of mutable types in Python.

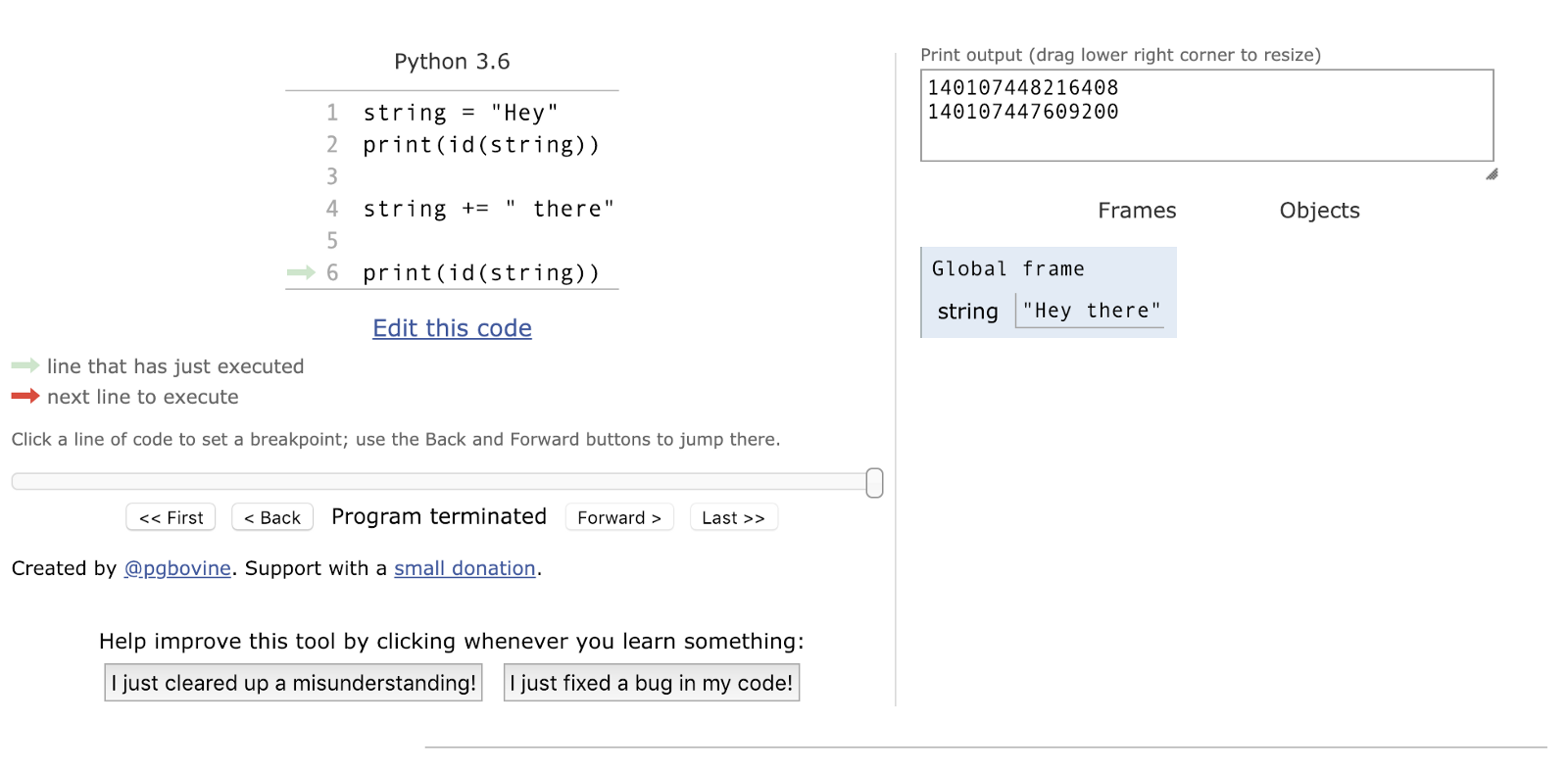
### Immutable Types

Some of Python’s include:

* integers (ex: x = 3)
* floats (ex: x = 3.14)
* strings (ex: x = “Hello”)
* tuples (ex: x = (20, 4))
* ranges (ex: range(6)
* booleans (ex: x = True)

Immutable types do not support modifications, hence the name immutable. So, when immutable types are changed (string concatenation for example) Python will return a new instance of a string object to be assigned to a variable name.

Link to demo: <https://goo.gl/F4z3nM>



Result from [PythonTutor.com](http://pythontutor.com/)

In conclusion, id(string) changes after we concatenate our string due to the immutable nature of the object itself. If it were in fact mutable, id(string)wouldn’t be a different number.

### Pay Attention Here

This is a huge deal because Python treats mutable and immutable objects differently. If you don’t keep track of the qualities of both, it might lead you to unexpected bugs. Take the following example:

>>> account\_balance = 1000000  
>>> balance\_to\_transfer = account\_balance  
>>> account\_balance = 2000000  
>>> balance\_to\_transfer  
1000000

Obviously banking systems aren’t this simple but you get the point… if you don’t know the types you are working with you could make some serious errors in your applications that could lead to being costly.

To make things more confusing take this example for instance. Note: remember that lists are mutable objects whereas integers (above example) are not.

Let’s say we have some software system in a grocery store that keeps track of two distinct lists of SKUs new\_items and perished\_items:

* sts for object identity which returns True if they are in fact the **same object**
* == is different though because it is testing the **value**of the two objects!

<http://openbookproject.net/thinkcs/python/english3e/tuples.html>

# 9. Tuples

In tuple packing, the values on the left are ‘packed’ together in a tuple:

**>>>** b = ("Bob", 19, "CS") *# tuple packing*

## In tuple unpacking, the values in a tuple on the right are ‘unpacked’ into the variables/names on the right

**>>>** b = ("Bob", 19, "CS")

**>>>** (name, age, studies) = b *# tuple unpacking*

**>>>** name

'Bob'

**>>>** age

19

**>>>** studies

'CS'

## 9.1. Tuples are used for grouping data

We saw earlier that we could group together pairs of values by surrounding with parentheses. Recall this example:

**>>>** year\_born = ("Paris Hilton", 1981)

This is an example of a **data structure** — a mechanism for grouping and organizing data to make it easier to use.

The pair is an example of a **tuple**. Generalizing this, a tuple can be used to group any [number](http://openbookproject.net/thinkcs/python/english3e/tuples.html) of items into a single compound value. Syntactically, a tuple is a comma-separated sequence of values. Although it is not necessary, it is conventional to enclose tuples in parentheses: