Introduction to programming using Python

Session 6

Matthieu Choplin

http://pythonguy.com/

Objectives

- To come back on the notion of object and type.
- To introduce to the type "List" and its methods.
- To use the len, min/max, sum, and random.shuffle functions for a list.
- To develop and invoke functions with list arguments and return value.
- To access list elements using indexed variables.
- To obtain a sublist using the slicing operator [start:end].
- To use +, *, and in/not in operators on lists.
- To traverse elements in a list using a for-each loop.
- To create lists using list comprehension.
- To split a string to a list using the str’s split method.
- To copy contents from one list to another.

What is the difference between an object and a type?

A **type** or a **class** is what is going to create an object

Built in types and objects seen so far:

Types	**Objects**	**Constructor**
Integer	1, 3, 4, 5, 999, -3, -4	int()
Float	1.333, -0.5, 0.001	float()
String	"Foo", 'bar', ""	str()

An object has methods

You can find the method of an object with the function **dir()**, which returns the attributes of an object.

```
>>> dir("abc")
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__
```

NB: the dunder methods (with double underscore), are "special methods" in python that can be overridden. We will come back on that later.

Difference between methods of objects and builtin functions

The methods of an object can **only be called on an object**.

```
>>> "speak louder".upper()
'SPEAK LOUDER'
```

A **builtin function** does not need an object to be called.

```
>>> len("number of character")
19
```

NB: **len()** give the number of element in a sequence

The type List

Creating list using the list constructor

```
list1 = list() # Create an empty list
list2 = list([2, 3, 4]) # Create a list with elements 2, 3, 4
list3 = list(["red", "green", "blue"]) # Create a list of strings
list4 = list(range(3, 6)) # Create a list with elements 3, 4, 5
list5 = list("abcd") # Create a list with characters a, b, c
```

That is the equivalent of:

```
list1 = [] # Same as list()
list2 = [2, 3, 4] # Same as list([2, 3, 4])
list3 = ["red", "green"] # Same as list(["red", "green"])
```

The List methods

You can find the different methods of a list thanks to the function **dir()**

```
>>> dir([])
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__'
```

We are going to look at: 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort'

How to see what a method can do

Look at the builtin help:

```
>>> help([].append)
Help on built-in function append:

append(...) method of builtins.list instance
    L.append(object) -> None -- append object to end
```

Experiment in the interpreter:

```
my_list = ["foo", "bar", 124, []]
print(my_list)
my_list.append("abc")
print(my_list)
```

Summary of the list methods

append(x: object): None	Add an item x to the end of the list.
insert(index: int, x: object): None	Insert an item x at a given index. Note that the first element in the list has index 0.
remove(x: object): None	Remove the first occurrence of the item x from the list.
index(x: object): int	Return the index of the item x in the list.
count(x: object): int	Return the number of times item x appears in the list.
sort(): None	Sort the items in the list.
reverse(): None	Reverse the items in the list.
extend(L: list): None	Append all the items in list L to the list.
pop([i]): object	Remove the item at the given position and return it. The square bracket denotes that parameter is optional. If no index is specified, list.pop() removes and returns the last item in the list.

Exercise

Write a program that reads integers from the user and stores them in a list (use input() and append()). Your program should continue reading values until the user enters 'q' (the sentinel value). Then it should display all of the values entered by the user in order from smallest to largest, with one value appearing on each line. Use either the sort method or the sorted built in function to sort the list.

👁 Solution

Builtin function for list or sequences

```
>>> list1 = [2, 3, 4, 1, 32]
>>> len(list1)
5
>>> max(list1)
32
>>> min(list1)
1
>>> sum(list1)
42
>>> import random
>>> random.shuffle(list1) # Shuffle the items in the list
>>> list1
[4, 1, 2, 32, 3]
```

Iterating on a list

The list is a **sequence** on which you can iterate.

With **for**:

```
for element in ["foo", 11, "bar"]:
    print(element)
```

With **while**:

```
my_list = ["foo", 11, "bar"]
i=0
while i < len(my_list):
  print(my_list[i])
  i+=1
```

Reminder about functions

We define the function like this:

```
def main():
    print('The function', main.__name__, 'has been called')
```

And we call the functions like this:

```
main()
```

NB: notice the brackets: when we define and when we call!

Try to use functions in the next exercises.

Passing Lists to Functions

```python
def printList(lst):
    for element in lst:
        print(element)

# Invoke the function
lst = [3, 1, 2, 6, 4, 2]
printList(lst)
```

Returning a List from a Function

Example: a function that returns a reversed list

```
def reverse(lst):
    result = []
    for element in lst:
        result.insert(0, element)
    return result

list1 = [1, 2, 3, 4, 5, 6]
list2 = reverse(list1)
print(list2)
```

The function **reverse** actually exists for doing the same thing

Exercise

Complete this program to get the minimum number of the list and its index

```
import random
random_list = [random.choice(list(range(1, 100))) for _ in range(10)]
def get_min(random_list):
    # to complete
    pass
get_min(random_list)
```

Solution without using built in functions non list methods

👁 Solution

Solution using built in functions and list methods

- 👁 Solution

Reminder

The string is a **sequence**

The items of a sequence can be **accessed** through indexes

Items (characters)	a	b	r	a	c	a	d	a	b	r	a
Indexes	0	1	2	3	4	5	6	7	8	9	10

Get the first element of the sequence:

```
my_string_variable = "abracadabra"
first_elem = my_string_variable[0]
```

Manipulate element of a List with indexes

You can also **access** element of a list with indexes **BUT** you can also **modify** them:

```
my_list = ["foo", 11, "bar"]
print(my_list)
my_list[1] = "eleven"
print(my_list)
```

contrary to the string type.

```
my_string = "abracadabra"
my_string[0] = "O"
print(my_string)
```

Difference between mutable and immutable objects

- You cannot modify an **immutable** object such as a string.
- You can modify a **mutable** object such as a list.

The +, *, [:], and in Operators (1/2)

+ is for concatenating list

***** is for repeating a list

[:] is the slice operator, for extracting a sublist from a list

```
>>> list1 = [2, 3]
>>> list2 = [1, 9]
>>> list3 = list1 + list2
>>> list3
[2, 3, 1, 9]
>>> list3 = 2 * list1
>>> list3
[2, 3, 2, 3]
>>> list4 = list3[2:4]
>>> list4
[2, 3]
```

The +, *, [:], and in Operators (2/2)

- Get the last element of a list with a negative index
- Check if an element is in a list with the **in** operator

```
>>> list1 = [2, 3, 5, 2, 33, 21]
>>> list1[-1]
21
>>> list1[-3]
2
>>> list1 = [2, 3, 5, 2, 33, 21]
>>> 2 in list1
True
>>> list1 = [2, 3, 5, 2, 33, 21]
>>> 2.5 in list1
False
```

List comprehensions

- List comprehensions provide a concise way to create lists
  - Transforming a list with operation on each element
  - Filtering a list, keeping only elements that satisfy a condition

```
>>> list1 = [x for x in range(0, 5)]
>>> list1
[0, 1, 2, 3, 4]
>>> list2 = [0.5 * x for x in list1]
>>> list2
[0.0, 0.5, 1.0, 1.5, 2.0]
>>> list3 = [x for x in list2 if x < 1.5]
>>> list3
[0.0, 0.5, 1.0]
```

Splitting a String to a List

You can convert a string to a list with the **split** function on string.

```
>>> items = "Welcome to the UK".split()
>>> print(items)
['Welcome', 'to', 'the', 'UK']
>>> items = "34#13#78#45".split("#")
>>> print(items)
['34', '13', '78', '45']
```

You can convert back a list to a string with the **join** function on string

```
>>> print(items)
['Welcome', 'to', 'the', 'UK']
>>> print(" ".join(items))
'Welcome to the UK'
```

Exercise - Eliminate duplicates

Write a function that returns a new list by eliminating the duplicate values in the list. Use the following function header:

```
def eliminateDuplicates(lst):
```

Write a test program that reads in a list of integers, invokes the function, and displays the result. Here is the sample run of the program:

```
Enter ten numbers: 1 2 3 2 1 6 3 4 5 2
The distinct numbers are: 1 2 3 6 4 5
```

Solution

👁 Solution

Exercise = Anagrams

Write a function that checks whether two words are anagrams. Two words are anagrams if they contain the same letters. For example, silent and listen are anagrams. The header of the function is:

```
def isAnagram(s1, s2):
```

(Hint: Obtain two lists for the two strings. Sort the lists and check if two lists are identical.)

Write a test program that prompts the user to enter two strings and, if they are anagrams, displays is an anagram; otherwise, it displays is not an anagram.

Solution

- 👁 Solution

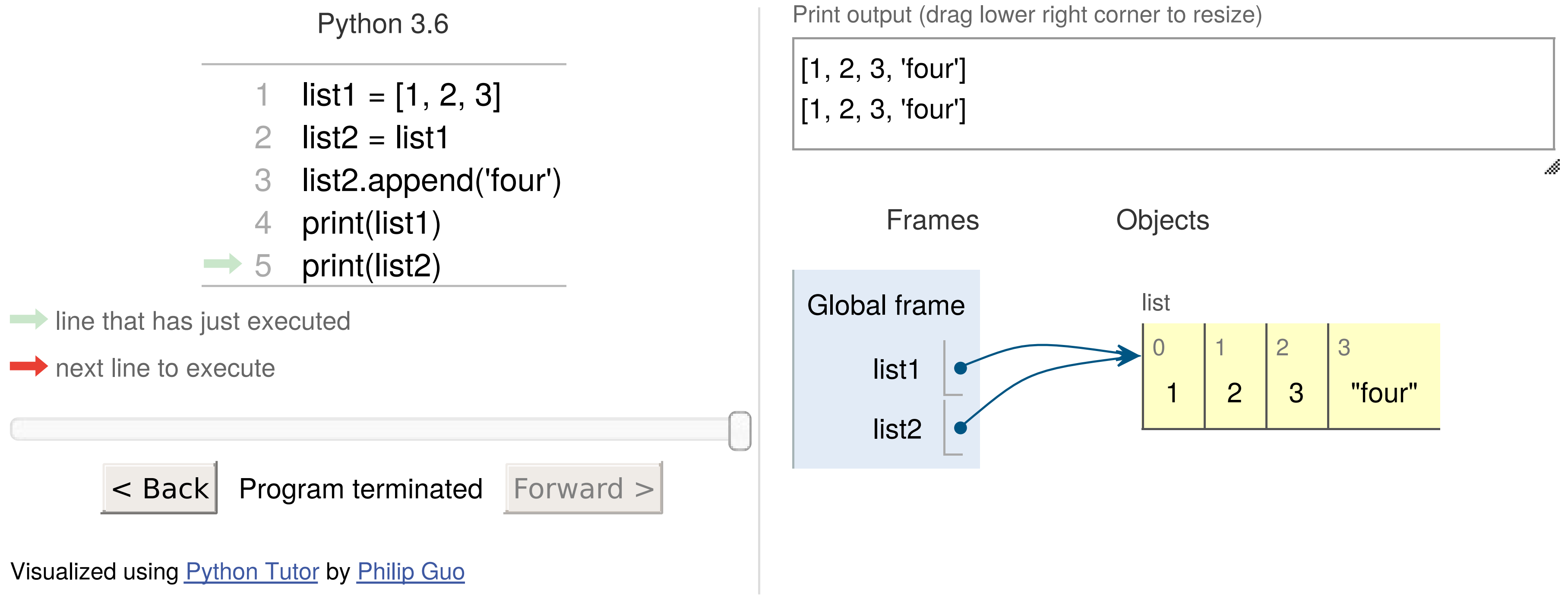
Copying Lists

Often, in a program, you need to duplicate a list or a part of a list. In such cases you could attempt to use the assignment statement (=):

```
list1 = [1, 2, 3]
list2=list1
```

But you are not copying the list here! You are copying its reference.

What is happening in memory



Python 3.6

```
list1 = [1, 2, 3]
list2 = list1
list2.append('four')
print(list1)
print(list2)
```

Visualized using Python Tutor by Philip Guo

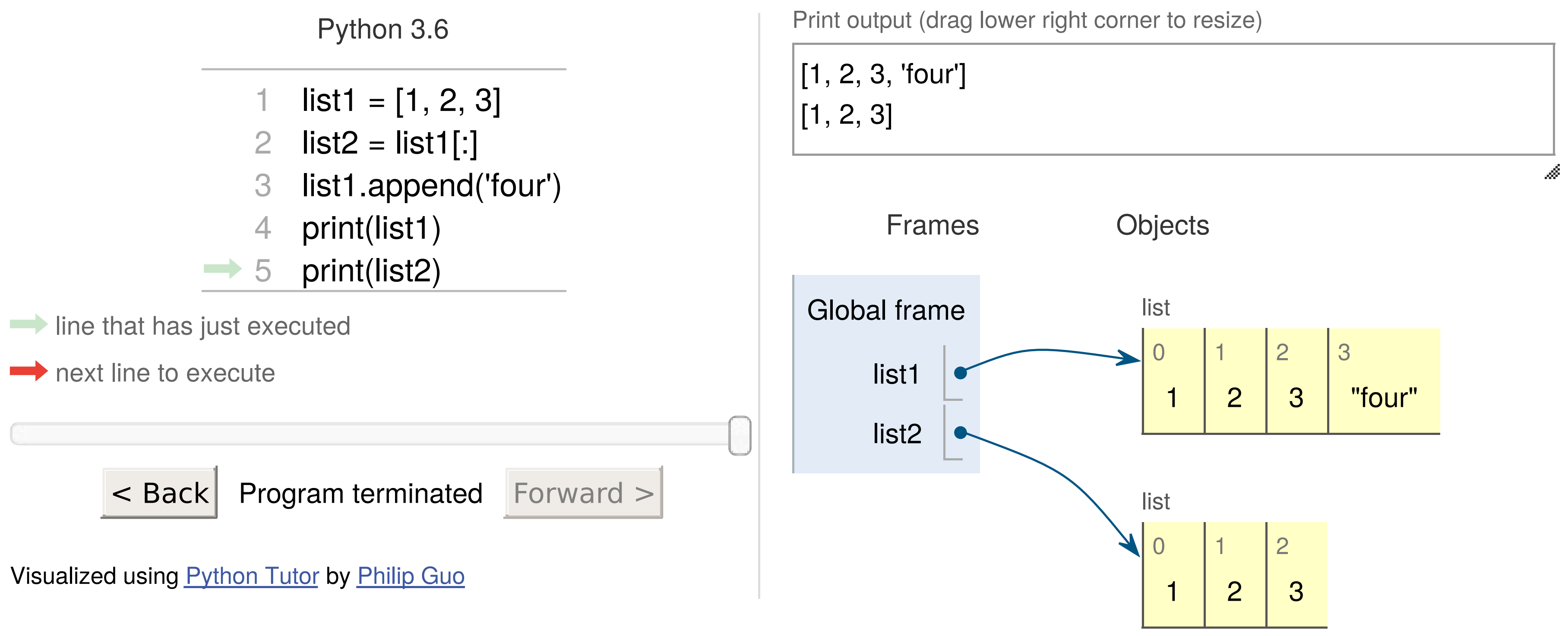
Print output (drag lower right corner to resize)

```
[1, 2, 3, 'four']
[1, 2, 3, 'four']
```

Copying a list the correct way

```
>>> list2 = [x for x in list1]
>>> list2 = list1[:]
>>> list2 = list(list1)
>>> list2 = list(list1)
>>> import copy
>>> list2 = copy.copy(list1)
>>> list2 = copy.deepcopy(list1) # will copy the object as well
```

What is happening in memory for a real copy



Python 3.6

```
list1 = [1, 2, 3]
list2 = list1[:]
list1.append('four')
print(list1)
print(list2)
```

➡ line that has just executed

➡ next line to execute

< Back Program terminated Forward >

Visualized using Python Tutor by Philip Guo

Print output (drag lower right corner to resize)

```
[1, 2, 3, 'four']
[1, 2, 3]
```

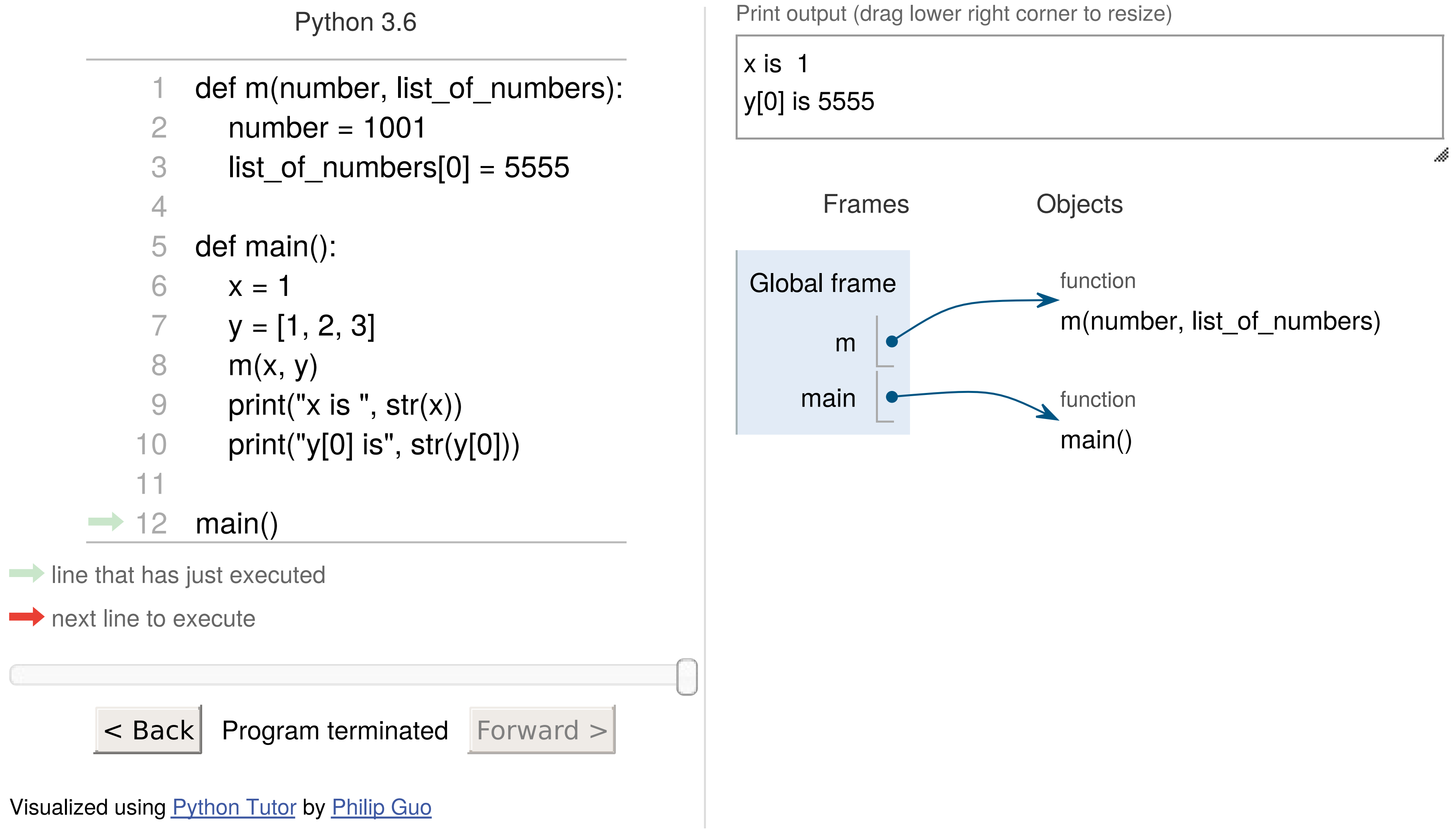
Pass By Value

There are important differences between passing immutable or mutable objects as arguments to a function.

String and numeric values (integer and float) are **immutable**, they do not get changed

Lists are **mutable**, they can be changed

Example



Python 3.6

```
def m(number, list_of_numbers):
    number = 1001
    list_of_numbers[0] = 5555

def main():
    x = 1
    y = [1, 2, 3]
    m(x, y)
    print("x is ", str(x))
    print("y[0] is", str(y[0]))

main()
```

Visualized using Python Tutor by Philip Guo

Print output (drag lower right corner to resize)

```
x is  1
y[0] is 5555
```