

Introduction to programming with Python

Session 4

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?

We use objects and types every day. Let's try to understand the difference with an every day example.

If I tell you: "Today I bought a pen. What colour is it?".
Is it **possible** to answer this question?

You already know what a pen is. It has specific properties:

- Colour
- Amount of ink
- Type (ballpoint, white board etc.)

What is the difference between an object and a type?

You also know that you can **write** with a pen.

So the question is not wrong. **Every** pen has a colour.

We have information about all pens, but we don't have information about the specific pen bought.

What **colour** is it? What **type** of pen is it?

If we see the pen all these questions are immediately solved.
It is clearly a blue, ballpoint pen.



What is the difference between an object and a type?

Every pen has these properties:

- Colour
- Amount of ink
- Type

and there is an action we can do:

- Write

This set of properties and actions describes the **idea** of a pen.



This pen has specific values for each of the properties that every pen has:

- Colour: **Blue**
- Amount of ink: **100%**
- Type: **Ballpoint pen**

Each real life pen **object** has different values for each of the properties.

What is the difference between an object and a type?

In Python, an idea is called a **type** or a **class**.

Exactly like in real life, it is possible to create **objects** from a type. You can create an infinite amount of objects from each type.

Type 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__',
```

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**, using values of its properties to produce a result.

```
>>> "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 a list object using the list constructor

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

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__',
```

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

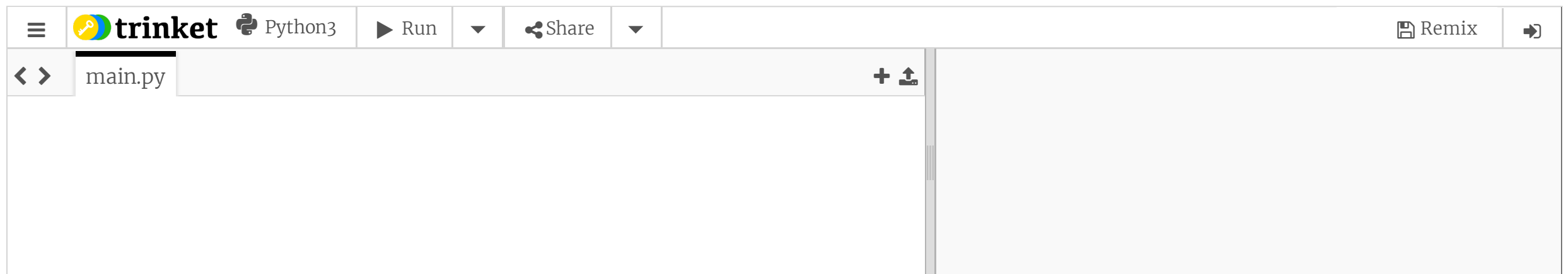
How to see what a method can do

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:



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 1

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

```
data = []
num = input("Enter an integer ('q' to quit): ")
while num != 'q':
    data.append(int(num))
    num = input("Enter an integer ('q' to quit): ")
data.sort()
print("The values, sorted into ascending order are:")
for element in data:
    print(element)
```

Built-in functions 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

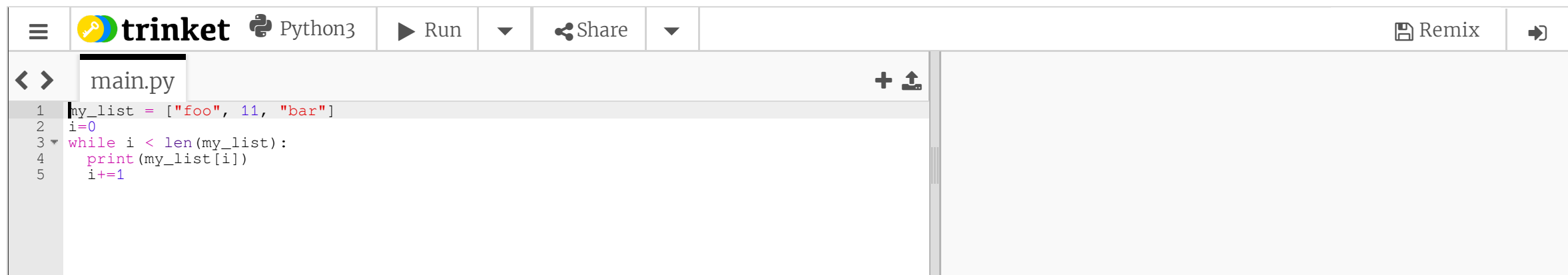
The list is a **sequence** on which you can iterate.

With **for**:



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

With **while**:



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


Reminder about functions

We define a function like this:

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

And we call a function like this:

```
main()
```

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

Try to use functions in the next exercises.

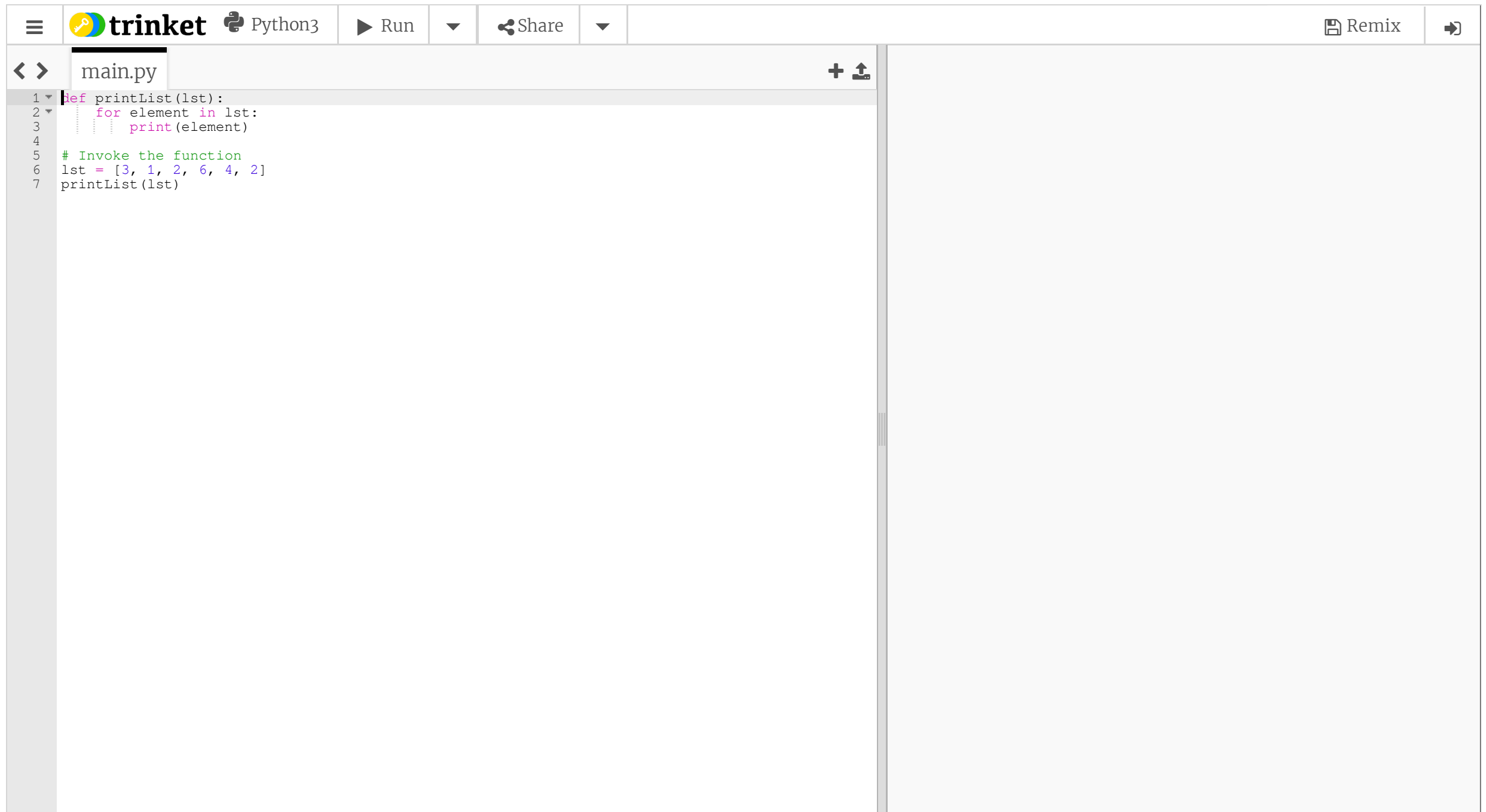
Exercise 2: Chinese Zodiac sign with list

Simplify the exercise we saw at the end of session 2 by using a list of `strings` storing all the animals names, instead of using multiple `if` and `elif` statements.

Solution

```
def main():  
    year = int(input("Enter a year: "))  
    animals = ["monkey", "rooster", "dog", "pig", "rat", "ox",  
               "tiger", "rabbit", "dragon", "snake", "horse", "sheep"]  
    print(year, "is", animals[year % 12])  
  
main()
```

Passing Lists to Functions

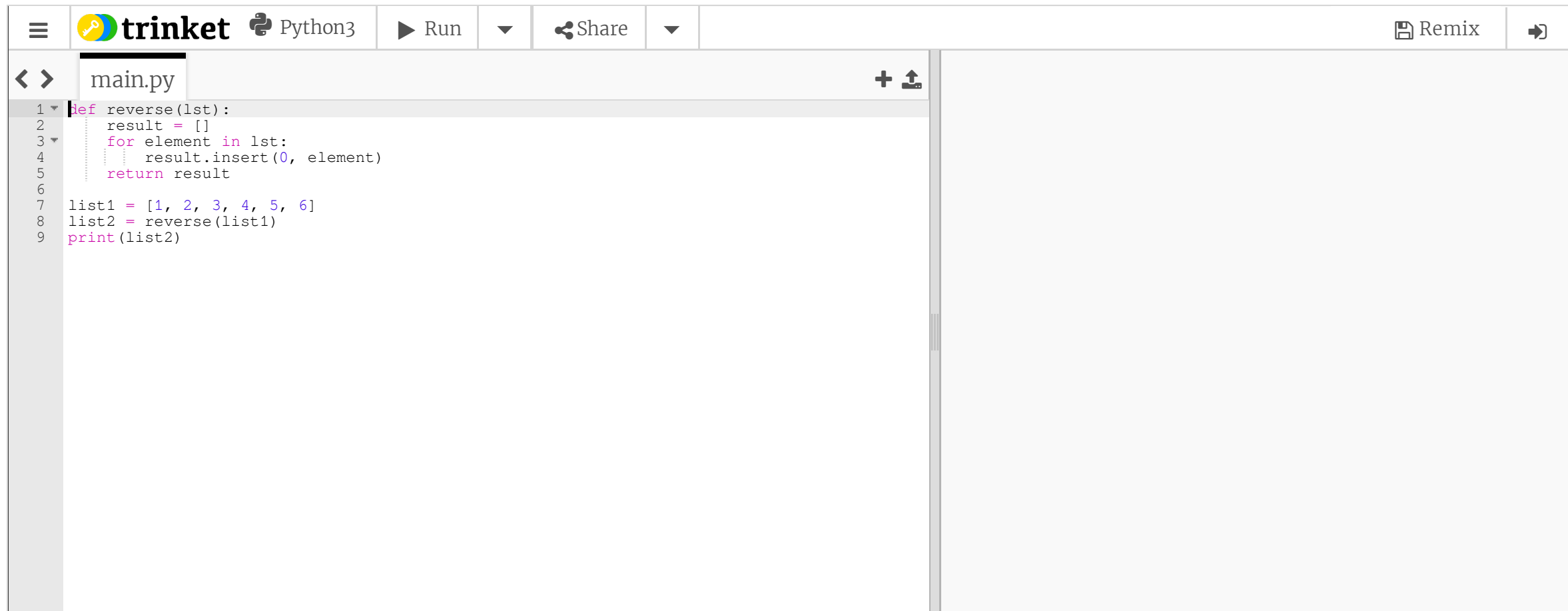


The image shows a screenshot of the Trinket Python3 IDE. The interface includes a top bar with the Trinket logo, Python3 interpreter, and buttons for Run, Share, and Remix. The main editor area displays a Python script named main.py. The script defines a function printList(lst) that iterates over the elements of the list and prints them. It then creates a list lst = [3, 1, 2, 6, 4, 2] and calls the printList function with this list as an argument.

```
1 def printList(lst):  
2     for element in lst:  
3         print(element)  
4  
5 # Invoke the function  
6 lst = [3, 1, 2, 6, 4, 2]  
7 printList(lst)
```

Returning a List from a Function

Example: a function that returns a reversed list



```
1 def reverse(lst):
2     result = []
3     for element in lst:
4         result.insert(0, element)
5     return result
6
7 list1 = [1, 2, 3, 4, 5, 6]
8 list2 = reverse(list1)
9 print(list2)
```

The **reverse** method of `list` works similarly, but only changes list values in place.

Exercise 3:

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 or list methods

Solution

```
import random
random_list = [random.choice(list(range(1, 100))) for _ in range(10)]
def get_min_index(any_list):
    min = 100
    index = 0
    for i in any_list:
        if i <= min:
            min = i
            min_index = index
        index += 1
    print("the min is", min)
    print("its index is", min_index)

print(random_list)
get_min_index(random_list)
```

Solution using built-in functions and list methods

Solution

```
import random
random_list = [random.choice(list(range(1, 100))) for _ in range(10)]
def get_min_index(any_list):
    print("the min is", min(any_list)) # using the min builtin
    print("its index is", random_list.index(min(random_list))) # u

print(random_list)
get_min_index(random_list)
```

Reminder

A string is a **sequence**

The items of a sequence can be **accessed** using indices

Items (characters)	a	b	r	a	c	a	d	a	b	r	a
-----------------------	---	---	---	---	---	---	---	---	---	---	---

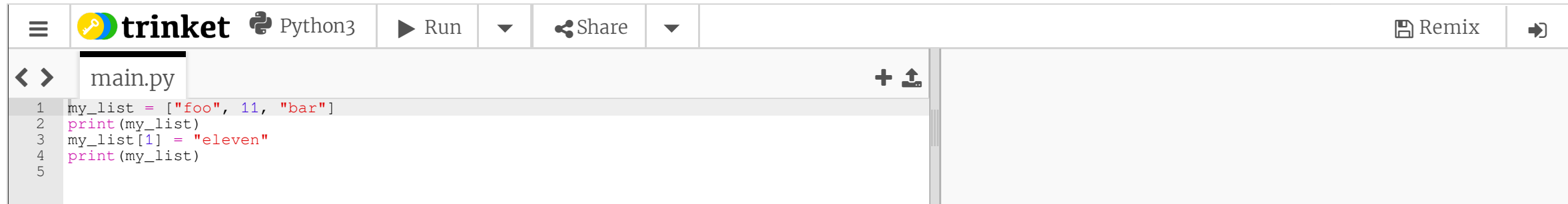
Indexes	0	1	2	3	4	5	6	7	8	9	10
---------	---	---	---	---	---	---	---	---	---	---	----

Get the first element of the sequence:

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

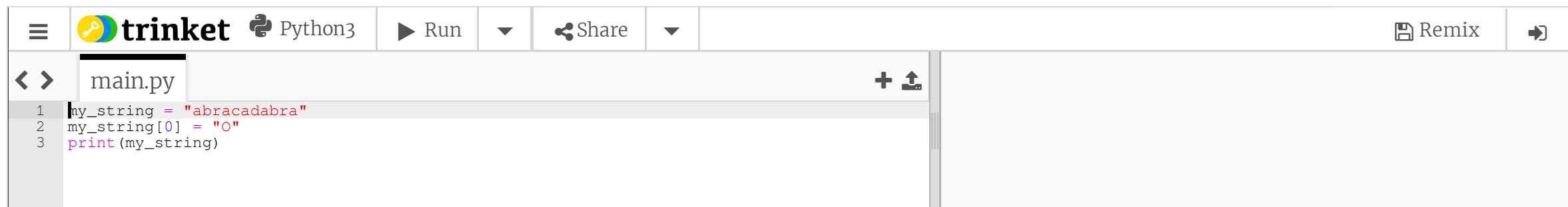

Manipulate List elements with indices

You can **access** and **modify** list elements using indices:



```
1 my_list = ["foo", 11, "bar"]
2 print(my_list)
3 my_list[1] = "eleven"
4 print(my_list)
5
```

You cannot modify a string using indices.



```
1 my_string = "abracadabra"
2 my_string[0] = "0"
3 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 Python course".split()
>>> print(items)
['Welcome', 'to', 'the', 'Python', 'course']
>>> 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', 'Python', 'course']
>>> print(" ".join(items))
'Welcome to the Python course'
```

Exercise 4 - 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

```
def main():
    # Read numbers as a string from the console
    s = input("Enter numbers: ")
    items = s.split() # Extracts items from the string
    numbers = [ int(x) for x in items ] # Convert items to numbers

    print("The distinct numbers are:", eliminateDuplicates(numbers))

def eliminateDuplicates(list):
    result = []
    for element in list:
        if not (element in result):
            result.append(element)

    return result

main()
```


Exercise 5: 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 the 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

```
def main():
    s1 = input("Enter the first string: ").strip()
    s2 = input("Enter the second string: ").strip()

    print(s1, "and", s2, "are",
          ("anagram." if isAnagram(s1, s2) else "not anagram."))

def isAnagram(s1, s2):
    if len(s1) != len(s2):
        return False

    newS1 = sort(s1);
    newS2 = sort(s2);

    return newS1 == newS2

def sort(s):
    r = list(s)
    r.sort()

    result = ""
    for ch in r:
        result += ch

    return result

main()
```

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

```
1 list1 = [1, 2, 3]
2 list2 = list1
3 list2.append('four')
4 print(list1)
→ 5 print(list2)
```

→ line that has just executed
→ next line to execute

< Back Program terminated Forward >

[Python Tutor](#) by [Philip Guo](#). Support with a [small donation](#).

Print output (drag lower right corner to resize)

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

Frames Objects

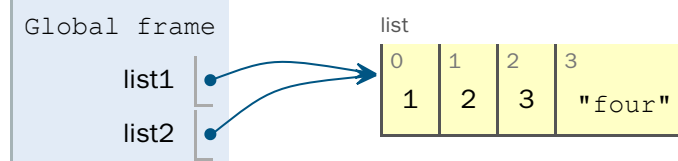
Global frame

list1

list2

list

0	1	2	3
1	2	3	"four"



The diagram illustrates the memory state. A 'Global frame' contains two variables, 'list1' and 'list2'. Both variables have arrows pointing to a single 'list' object. This object is represented as an array with four elements: 1, 2, 3, and 'four'. The indices 0, 1, 2, and 3 are shown above the elements. The 'list' object is highlighted in yellow.

Correctly copying a list

```
>>> 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

```
1 list1 = [1, 2, 3]
2 list2 = list1[:]
3 list1.append('four')
4 print(list1)
→ 5 print(list2)
```

→ line that has just executed
→ next line to execute

< Back Program terminated Forward >

[Python Tutor](#) by [Philip Guo](#). Support with a [small donation](#).

Print output (drag lower right corner to resize)

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

Frames

Global frame

list1

list2

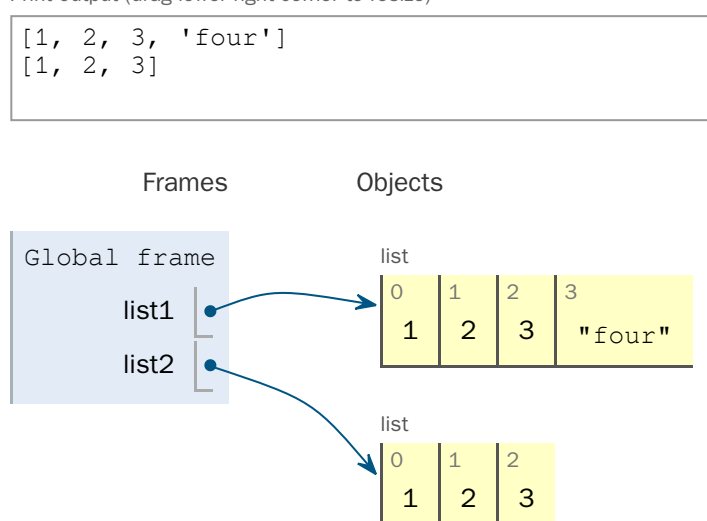
Objects

list

0	1	2	3
1	2	3	"four"

list

0	1	2
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

```
1 def m(number, list_of_numbers):
2     number = 1001
3     list_of_numbers[0] = 5555
4
5 def main():
6     x = 1
7     y = [1, 2, 3]
8     m(x, y)
9     print("x is ", str(x))
10    print("y[0] is", str(y[0]))
11
12 main()
```

→ line that has just executed
→ next line to execute

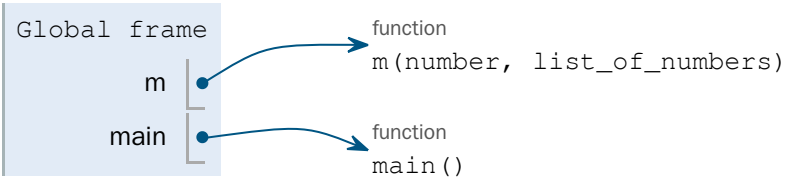
< Back Program terminated Forward >

[Python Tutor](#) by [Philip Guo](#). Support with a [small donation](#).

Print output (drag lower right corner to resize)

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

Frames Objects



```
graph LR
    subgraph Frames
        GF[Global frame]
        GF -- contains --> m_var[m]
        GF -- contains --> main_var[main]
    end
    subgraph Objects
        m_obj["function m(number, list_of_numbers)"]
        main_obj["function main()"]
    end
    m_var -- points to --> m_obj
    main_var -- points to --> main_obj
```


Exercise 6: Hangman

Write a hangman game that randomly generates a word and prompts the user to guess one letter at a time, as shown in the sample run.

Each letter in the word is displayed as an asterisk. When the user makes a correct guess, the actual letter is then displayed. When the user finishes a word, display the number of misses and ask the user whether to continue playing. Create a list to store the words, as follows:

```
words = ["write", "that", "program", ...]
```

```
(Guess) Enter a letter in word ***** > p
(Guess) Enter a letter in word p***** > r
(Guess) Enter a letter in word pr**r** > p
    p is already in the word
(Guess) Enter a letter in word pr**r** > o
(Guess) Enter a letter in word pro*r** > g
(Guess) Enter a letter in word progr** > n
    n is not in the word
(Guess) Enter a letter in word progr** > m
(Guess) Enter a letter in word progr*m > a
The word is program. You missed 1 time
Do you want to guess another word? Enter y or n>
```

```
import random

def main():
    words = ["write", "program", "that", "receive", "positive", "change", "study", "excellent", "n
gameFinished = False
while not gameFinished:
    index = random.randint(0, len(words) - 1)
    hiddenWord = words[index]
    guessedWord = len(hiddenWord) * ['*']
    numberOfCorrectLettersGuessed = 0
    numberOfMisses = 0
    while numberOfCorrectLettersGuessed < len(hiddenWord):
        letter = input("(Guess) Enter a letter in word " + toString(guessedWord) + " > ").stri
        if letter in guessedWord:
            print("\t", letter, "is already in the word")
        elif hiddenWord.find(letter) < 0:
            print("\t", letter, "is not in the word")
            numberOfMisses += 1
        else:
            k = hiddenWord.find(letter)
            while k >= 0:
                guessedWord[k] = letter
                numberOfCorrectLettersGuessed += 1
                k = hiddenWord.find(letter, k + 1)
    print("The word is " + hiddenWord + ". You missed "
          + str(numberOfMisses) + (" time" if (numberOfMisses <= 1) else " times"))
    anotherGame = input("Do you want to guess for another word? Enter y or n> ").strip()
    if anotherGame == 'n':
        print("Finished")
        gameFinished = True

def toString(list):
    s = ""
    for e in list:
        s += e
    return s

main()
```