PYTHON

YT0744

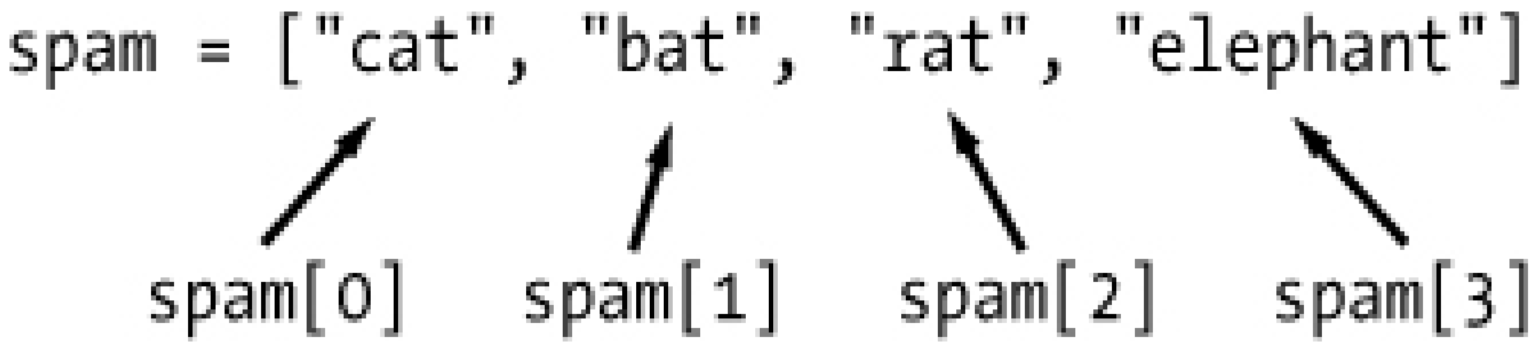
CHAPTER 4: LISTS

Johan Van Bauwel
E-mail: Johan.vanbauwel@thomasmore.be
Office: A114

THE LIST DATA TYPE

- **Definition:** A list is a collection of elements that contains multiple values in an ordered sequence. Lists are *mutable*, which means the list items can change value after being created.
- **Example:** spam = ['cat', 'bat', 'rat', 'elephant']
- Begins with opening square bracket and ends with closing square bracket [ ]
- Values inside list → items
- Items separated by commas

A LIST WITH INDEXES


```
spam = ["cat", "bat", "rat", "elephant"]
```

- **Index:** integer inside the square bracket that follows the list
- spam[0] evaluate to ‘cat’ and spam[1] to ‘bat’, …
- First value is at index 0, second at index 1, …

A LIST WITH INDEXES: EXAMPLES

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0]
'cat'
>>> spam[1]
'bat'
>>> spam[2]
'rat'
>>> spam[3]
'elephant‘
>>> ['cat', 'bat', 'rat', 'elephant'][3]
'elephant'
>>> 'Hello ' + spam[0]
'Hello cat'
>>> 'The ' + spam[1] + ' ate the ' + spam[0] + '.'
'The bat ate the cat.'
```

INDEXERROR

IndexError: error message if you use an index that *exceeds* the *number of values* in your *list value*

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[10000]
Traceback (most recent call last):
File "<pyshell#9>", line 1, in <module>
spam[10000]
IndexError: list index out of range
```

TYPEERROR

TypeError: when you use *floats instead of integers as indexes*, indexes must be integer values!

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1]
'bat'
>>> spam[1.0]
Traceback (most recent call last):
File "<pyshell#13>", line 1, in <module>
spam[1.0]
TypeError: list indices must be integers, not float
>>> spam[int(1.0)]
'bat'
```

MULTIPLE INDEXES

Lists can contain *other list values*, the values in these lists of lists can be accessed by using multiple indexes

- First index: choosing the list
- Second index: value within the chosen list
- Only one value: printing the whole list

```
>>> spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]
>>> spam[0]
['cat', 'bat']
>>> spam[0][1]
'bat'
>>> spam[1][4]
50
```

NEGATIVE INDEXES

-1 refers to the last index,
-2 refers to the second-to-last index, …

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[-1]
'elephant'
>>> spam[-3]
'bat'
>>> 'The ' + spam[-1] + ' is afraid of the ' + spam[-3] + '.'
'The elephant is afraid of the bat.'
```

GETTING SUBLISTS WITH SLICES

- **Slice:** get *several values* from a list in the *form of a new list*
- Typed between square brackets but with two integers seperated by a colon
- Difference with list:
  - spam[2] is a list with an index (one integer).
  - spam[1:4] is a list with a slice (two integers).
- First integer → index where slice starts
- Second integer → index where slice ends
- !!! Value of the second index is not included !!!

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0:4]
['cat', 'bat', 'rat', 'elephant']
>>> spam[1:3]
['bat', 'rat']
>>> spam[0:-1]
['cat', 'bat', 'rat']
```

LENGTH OF LIST AND CHANGING VALUES

len() function will return number of values in a list value

```
>>> spam = ['cat', 'dog', 'moose']
>>> len(spam)
3
```

Use an index of a list to change the value at that index

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1] = 'aardvark'
>>> spam
['cat', 'aardvark', 'rat', 'elephant']
>>> spam[2] = spam[1]
>>> spam
['cat', 'aardvark', 'aardvark', 'elephant']
>>> spam[-1] = 12345
>>> spam
['cat', 'aardvark', 'aardvark', 12345]
```

LIST CONCATENATION AND LIST REPLICATION

- The + operator combines two lists to create a new list
- The * operator (using an integer) is used to replicate a list

```
>>> [1, 2, 3] + ['A', 'B', 'C']
[1, 2, 3, 'A', 'B', 'C']
>>> ['X', 'Y', 'Z'] * 3
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
>>> spam = [1, 2, 3]
>>> spam = spam + ['A', 'B', 'C']
>>> spam
[1, 2, 3, 'A', 'B', 'C']
```

DEL STATEMENT

- The del statement is used to remove a value at certain index in a list
- After the item at an index is removed, the other items in that list will be moved up one index

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat']
```

WORKING WITH LISTS

- Useful if you need to store a lot of similar values
- Don’t need to create a single variable for each item
- Benefit of lists:
  - Data is in a structure
  - Program is more flexible
- Bad and good examples in the following 2 slides

BAD EXAMPLE (WITHOUT LISTS)

```
catName1 = 'Zophie'
catName2 = 'Pooka'
catName3 = 'Simon'
catName4 = 'Lady Macbeth'
catName5 = 'Fat-tail'
catName6 = 'Miss Cleo‘
print('Enter the name of cat 1:')
catName1 = input()
print('Enter the name of cat 2:')
catName2 = input()
print('Enter the name of cat 3:')
catName3 = input()
print('Enter the name of cat 4:')
catName4 = input()
print('Enter the name of cat 5:')
catName5 = input()
print('Enter the name of cat 6:')
catName6 = input()
print('The cat names are:')
print(catName1 + ' ' + catName2 + ' ' + catName3 + ' '
+ catName4 + ' ' + catName5 + ' ' + catName6)
```

- Lot of code duplication or identical code
- Not able to add more values to variables

GOOD EXAMPLE (WITH LISTS)

```
catNames = []
while True:
        print('Enter the name of cat ' + str(len(catNames) + 1) + ' (Or enter nothing to stop.):')
        name = input()
        if name == '':
                break
        catNames = catNames + [name] # list concatenation
print('The cat names are:')
for name in catNames:
        print(' ' + name)
```

```
Enter the name of cat 1 (Or enter nothing to stop.):
Zophie
Enter the name of cat 2 (Or enter nothing to stop.):
Pooka
Enter the name of cat 3 (Or enter nothing to stop.):
Simon
Enter the name of cat 4 (Or enter nothing to stop.):
Lady Macbeth
Enter the name of cat 5 (Or enter nothing to stop.):
Fat-tail
Enter the name of cat 6 (Or enter nothing to stop.):
Miss Cleo
Enter the name of cat 7 (Or enter nothing to stop.):
```

- No code duplication or identical code
- Can store any number of values

```
*
The cat names are:
 Zophie
 Pooka
 Simon
 Lady Macbeth
 Fat-tail
 Miss Cleo
```

USING FOR LOOPS WITH LISTS

- A common technique to iterate over all indexes in a list is to use range(len(*someList*))

```
>>> supplies = ['pens', 'staplers', 'flame-throwers', 'binders']
>>> for i in range(len(supplies)):
        print('Index ' + str(i) + ' in supplies is: ' + supplies[i])
Index 0 in supplies is: pens
Index 1 in supplies is: staplers
Index 2 in supplies is: flame-throwers
Index 3 in supplies is: binders
```

THE IN AND NOT IN OPERATORS

- To determine whether a value is or isn't in a list

```
>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas']
True
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> 'cat' in spam
False
>>> 'howdy' not in spam
False
>>> 'cat' not in spam
True
```

THE MULTIPLE ASSIGNMENT TRICK

- A shortcut to assign multiple variables with the values in a list in a single line of code
- Instead of doing this:

```
>>> cat = ['fat', 'black', 'loud']
>>> size = cat[0]
>>> color = cat[1]
>>> disposition = cat[2]
```

- You can type this:

```
>>> cat = ['fat', 'black', 'loud']
>>> size, color, disposition = cat
```

- ValueError when the number of variables is not equal to the number of items in the list

```
>>> cat = ['fat', 'black', 'loud']
>>> size, color, disposition, name = cat
Traceback (most recent call last):
File "<pyshell#84>", line 1, in <module>
size, color, disposition, name = cat
ValueError: need more than 3 values to unpack
```

AUGMENTED ASSIGNMENT OPERATORS

- A shorcut for some operators

Augmented assignment statement	Equivalent assignment statement
spam = spam + 1	spam +=1
spam = spam – 1	spam -=1
spam = spam * 1	spam *=1
spam = spam / 1	spam /=1
spam = spam % 1	spam %=1

```
>>> spam = 42
>>> spam = spam + 1
>>> spam
43
```

```
>>> spam = 42
>>> spam += 1
>>> spam
43
```

- The += operator → string and list concatenation
- The *= operator → string and list replication

METHODS

- Same thing as a function, except it is “called on” a value
- Method part comes after the value, seperated by a period
- Each data type → own set of methods
- List data type:
    - index() method: finding value in a list
    - append() and insert() method: adding values to a list
    - remove() method: removing values from a list
    - sort() method: sort items in a list

INDEX() METHOD

- Finding value in a list
- If value exists: index of value is returned
- If value doesn’t exist: ValueError error

```
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> spam.index('hello')
0
>>> spam.index('heyas')
3
>>> spam.index('howdy howdy howdy')
Traceback (most recent call last):
File "<pyshell#31>", line 1, in <module>
spam.index('howdy howdy howdy')
ValueError: 'howdy howdy howdy' is not in list
```

- Duplicate values in list: only first appearance returned

```
>>> spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']
>>> spam.index('Pooka')
1
```

APPEND() AND INSERT() METHOD

- *Append() method:* add a value to the end of a list

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.append('moose')
>>> spam
['cat', 'dog', 'bat', 'moose']
```

- *Insert() method:* add a value at the given index in a list
  - *First argument:* index
  - *Second argument:* new value

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.insert(1, 'chicken')
>>> spam
['cat', 'chicken', 'dog', 'bat']
```

REMOVE METHOD()

- Remove a value from a list

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('bat')
>>> spam
['cat', 'rat', 'elephant']
```

- Remove a value that isn’t in the list: ValueError error

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('chicken')
Traceback (most recent call last):
File "<pyshell#11>", line 1, in <module>
spam.remove('chicken')
ValueError: list.remove(x): x not in list
```

- Value multiple times in list: only first appearence deleted

```
>>> spam = ['cat', 'bat', 'rat', 'cat', 'hat', 'cat']
>>> spam.remove('cat')
>>> spam
['bat', 'rat', 'cat', 'hat', 'cat']
```

SORT METHOD()

- To sort a list of numbers or a list of strings

```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
[-7, 1, 2, 3.14, 5]
>>> spam = ['ants', 'cats', 'dogs', 'badgers',
'elephants']
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

- Pass True for the reverse keyword to sort in reverse order

```
>>> spam.sort(reverse=True)
>>> spam
['elephants', 'dogs', 'cats', 'badgers', 'ants']
```

SORT METHOD(): NOTES

- List with both numbers and string cannot be sorted

```
>>> spam = [1, 3, 2, 4, 'Alice', 'Bob']
>>> spam.sort()
Traceback (most recent call last):
File "<pyshell#70>", line 1, in <module>
spam.sort()
TypeError: unorderable types: str() < int()
```

- Sort() use ASCIIbetical order: uppercase before lowercase

```
>>> spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
>>> spam.sort()
>>> spam
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
```

- Alphabetical order: string.lower for the key keyword → all items treated as lowercase items without changing the list

```
>>> spam = ['a', 'z', 'A', 'Z']
>>> spam.sort(key=str.lower)
>>> spam
['a', 'A', 'z', 'Z']
```

LIST-LIKE TYPES: STRINGS

- Strings are simalar to lists
- String is a ‘list’ of single text characters
- Many operations of lists can be done on strings
- Important difference between lists and strings
  - List are mutable → can change
  - Strings are inmutable → can’t change

```
>>> name = 'Zophie'
>>> name[0]
'Z'
>>> name[-2]
'i'
>>> name[0:4]
'Zoph'
>>> 'Zo' in name
True
>>> 'z' in name
False
>>> 'p' not in name
False
```

```
>>> for i in name:
print('* * * ' + i + ' * * *')
* * * Z * * *
* * * o * * *
* * * p * * *
* * * h * * *
* * * i * * *
* * * e * * *
```

```
>>> name = 'Zophie a cat'
>>> newName = name[0:7] + 'the' + name[8:12]
>>> name
'Zophie a cat'
Lists 95
>>> newName
'Zophie the cat'
```

```
>>> eggs = [1, 2, 3]
>>> eggs = [4, 5, 6]
>>> eggs
[4, 5, 6]
```

LIST-LIKE TYPES: TUPLES

- Almost identical to list data type
- Differences:
  - Tuples are typed with parentheses
  - Tuples are **immutable** like strings

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[0]
'hello'
>>> eggs[1:3]
(42, 0.5)
>>> len(eggs)
3
```

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[1] = 99
Traceback (most recent call last):
File "<pyshell#5>", line 1, in <module>
eggs[1] = 99
TypeError: 'tuple' object does not support item
assignment
```

  - One value in tuple → trailing comma after the value → otherwise, it's a regular value between strings

```
>>> type(('hello',))
<class 'tuple'>
>>> type(('hello'))
<class 'str'>
```

LISTS() AND TUPLE() FUNCTION

- list() → to convert data to a list
- tuple() → to convert data to a tuple

```
>>> tuple(['cat', 'dog', 5])
('cat', 'dog', 5)
>>> list(('cat', 'dog', 5))
['cat', 'dog', 5]
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
```

REFERENCES

- When you assign a *list to a variable*, you are actually assigning a *list reference* to the variable. A *reference* is a *value* that *points* to *some bit of data*, and a list reference is a value that points to a list.

```
>>> spam = [0, 1, 2, 3, 4, 5]
>>> cheese = spam
>>> cheese[1] = 'Hello!'
>>> spam
[0, 'Hello!', 2, 3, 4, 5]
>>> cheese
[0, 'Hello!', 2, 3, 4, 5]
```

- When you do this with strings or integers you will get this → because the variable contains the string or integer value

```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
>>> cheese
42
```

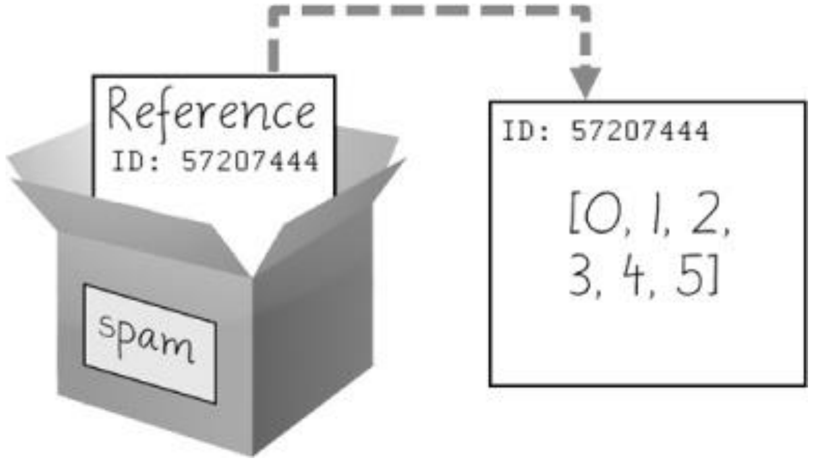
REFERENCES: EXPLANATION

1) Creating a list → reference to list in spam variable
2) Copies list reference in spam to cheese, not the list value
   - Values in spam and cheese → refer to same list
   - Only one underlying list → list never copied
3) Modify first element of cheese
   - → actually modifying same list that spam refers to

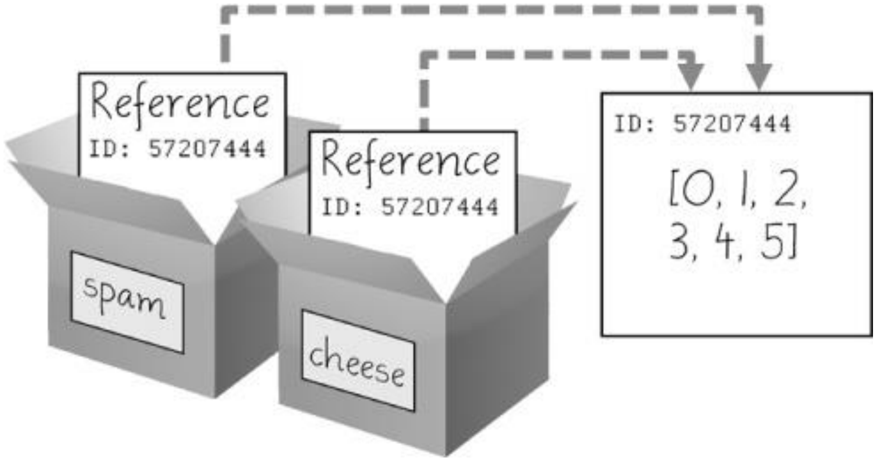
```
1)      >>> spam = [0, 1, 2, 3, 4, 5]
2)      >>> cheese = spam
3)      >>> cheese[1] = 'Hello!'
        >>> spam
        [0, 'Hello!', 2, 3, 4, 5]
        >>> cheese
        [0, 'Hello!', 2, 3, 4, 5]
```

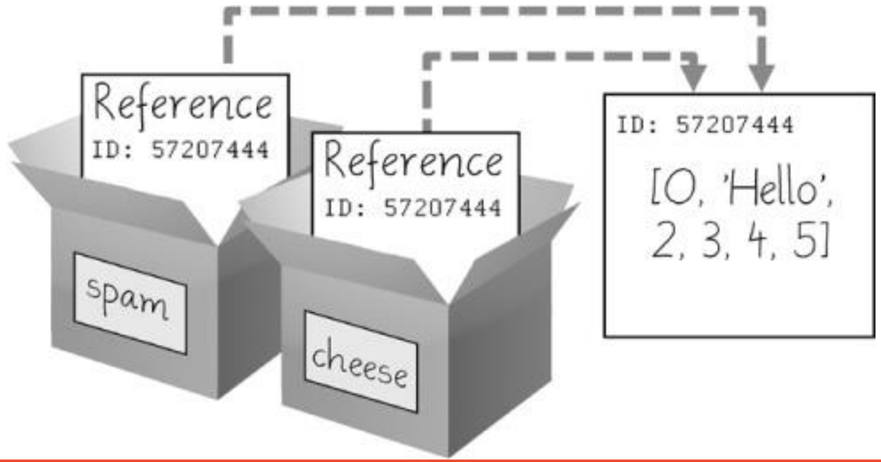
REFERENCES: EXPLANATION IN PICTURES

spam = [0, 1, 2, 3, 4, 5] stores a reference to a list, not the actual list.



spam = cheese copies the reference, not the list.





cheese[1] = 'Hello!' modifies the list that both variables refer to.

PASSING REFERENCES

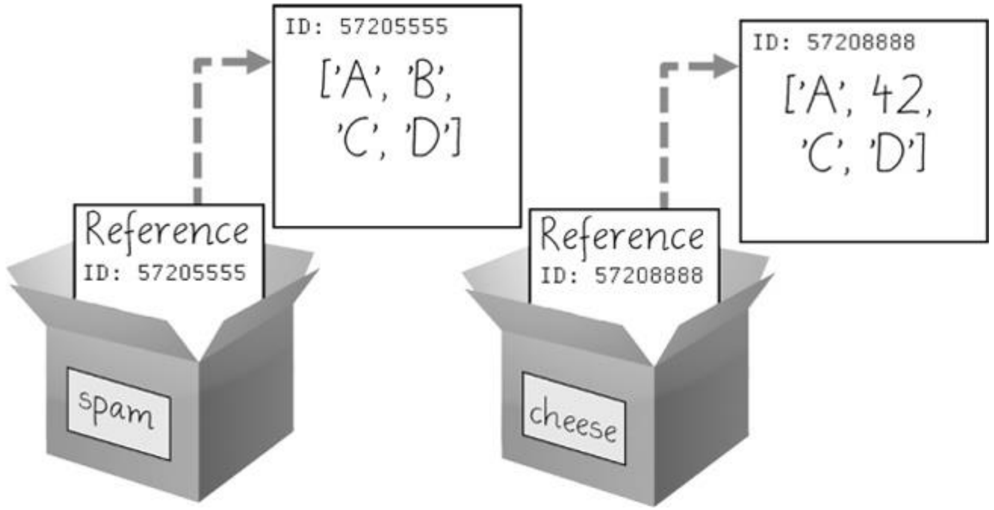
- Very important for understanding how arguments are passed to a function
- Values of argument are copied to the parameter variables
  → for lists this means the reference is copied
- List is modified directly in function
- !!! Important to remember → prevent confusing bugs !!!

```
def eggs(someParameter):
        someParameter.append('Hello')
spam = [1, 2, 3]
eggs(spam)
print(spam)
[1, 2, 3, 'Hello']
```

COPY() AND DEEPCOPY() FUNCTION

- If you don't want that functions change you list you can make a duplicate (you have to import the copy module)
  - Copy.copy() → duplicate copy of a mutable value
  - Copy.deepcopy() → if the list to copy contains lists
    → inner lists will be copied as well

```
>>> import copy
>>> spam = ['A', 'B', 'C', 'D']
>>> cheese = copy.copy(spam)
>>> cheese[1] = 42
>>> spam
['A', 'B', 'C', 'D']
>>> cheese
['A', 42, 'C', 'D']
```



cheese = copy.copy(spam) creates a second list that can be modified independently of the first.

SUMMARY

- Lists are mutable → contents can change
  - Append() function
  - Insert() function
  - Remove() function
- List like strings and tuples are immutable → can't change
- Variables do not store list values directly, but the references to lists
  - Important when you use functions → references are copied
  - Function changes original list
- Function copy() and deepcopy()
  - Changes lists in one variable → original list unchanged

Q & A

- Questions?