



Lists



Outline

- Lists
 - Creating a list
 - Indexing and slicing
 - List manipulation
 - Change elements
 - List methods: append, insert, remove, extend, sort, count, pop
 - Operators: del, in, +, *
 - Build-in function: len, max, min, sum
 - Lists and strings
 - split, join
- Class and object
- Python variable and memory allocation

Creating a list

- Like a string, a list is a **sequence of values**.
 - In a string, the values are characters; in a list, they can be **any type**.
- The values in list are called elements or items.

string	h	e	l	l	o		w	o	r	l	d	→ character
	0	1	2	3	4	5	6	7	8	9	10	→ Index
list	'Leo'		'Nora'		'Emma'		'James'		'Lucy'			→ item
	0		1		2		3		4			→ Index

- The simplest way to create a list is to enclose the elements in square brackets.

```
name_list = ["Leo", "Nora", "Emma", "James", "Lucy"]
```

Creating a list: Types of elements

- In a list, the elements can be any type.

An empty list

```
empty_list = []
```

A list of numbers

```
price_list = [100, 200, 550, 300, 450, 150, 200]
```

A list of lists

```
birthday_ddmm = [[1,10],[16,4],[19,2],[11,12],[9,7]]
```

- The elements of a list don't have to be the same type.

A list contains a string, a float, an integer, and another list

```
info = ['spam', 2.0, 5, [10, 20]]
```

Indexing and slicing

- Because lists are sequences, indexing and slicing work the same way for lists as they do for strings.

'Leo'	'Nora'	'Emma'	'James'	'Lucy'	→ item
0	1	2	3	4	→ Index

```
name_list = ["Leo", "Nora", "Emma", "James", "Lucy"]
```

```
print(name_list[1])
```

Nora

```
print(name_list[0:2])
```

```
['Leo', 'Nora']
```

↓ From position 0 to position 2, but NOT including 2.

'Leo'	'Nora'	'Emma'	'James'	'Lucy'
0	1	2	3	4

Indexing and slicing

The format for list slicing: [**start_index** : **stop_index**]

```
print(name_list[:2])
```

 If start_index is not specified, the slice starts at the beginning.
['Leo', 'Nora']

'Leo'	'Nora'	'Emma'	'James'	'Lucy'
0	1	2	3	4

```
print(name_list[3:])
```

 If end_index is not specified, the slice goes to the end.
['James', 'Lucy']

'Leo'	'Nora'	'Emma'	'James'	'Lucy'
0	1	2	3	4

```
print(name_list[:])
```

 If neither start_index nor stop_index is specified, the slice is a copy of the whole list.
['Leo', 'Nora', 'Emma', 'James', 'Lucy']

'Leo'	'Nora'	'Emma'	'James'	'Lucy'
0	1	2	3	4

Indexing and slicing - negative index

- Use negative indexes to start the slice from the end of the list.

'Leo'	'Nora'	'Emma'	'James'	'Lucy'
0	1	2	3	4
-5	-4	-3	-2	-1

→ Index

→ Negative Index

```
name_list = ["Leo", "Nora", "Emma", "James", "Lucy"]  
print(name_list[-1])
```

Lucy

```
print(name_list[-3:-1])
```

['Emma', 'James']

'Leo'	'Nora'	'Emma'	'James'	'Lucy'
-5	-4	-3	-2	-1

```
print(name_list[-3:])
```

['Emma', 'James', 'Lucy']

'Leo'	'Nora'	'Emma'	'James'	'Lucy'
-5	-4	-3	-2	-1

Indexing and slicing

- A list of lists

```
#[day,month]  
birthday_ddmm = [[12, 'Oct'], [16, 'Apr'], [19, 'Feb'], [11, 'Dec'], [9, 'Jul']]
```

12 Oct	16 Apr	19 Feb	11 Dec	9 Jul	→ item
0	1	2	3	4	→ Index

```
birthday_ddmm[0] #get the birthday of the first person
```

```
[12, 'Oct']
```

```
birthday_ddmm[0][1] #get birthday month of the first person
```

```
'Oct'
```

12	Oct	→ item
0	1	→ Index

Exercise

(A.1) Define a list named `str_list`, which contains the following elements: `A`, `B`, `C`, `D`, `E`. Print out the list.

(A.2) Print the first element in `str_list`.

(A.3) Use slice notation to print out `['B', 'C', 'D']`.

(A.4) Print the last element in `str_list`.

(A.5) Print the last two elements in `str_list`.

Outline

- Lists
 - Creating a list
 - Indexing and slicing
 - List manipulation
 - **Change elements**
 - **List methods:** append, insert, remove, extend, sort, count, pop
 - **Operators:** del, in, +, *
 - **Build-in function:** len, max, min, sum
 - Lists and strings
 - split, join
- Class and object
- Python variable and memory allocation

List manipulation (1) change elements

- Unlike strings, **lists are mutable**. When using a list, you can change its contents by assigning to either a particular item or an entire section (slice).
 - Index assignment

```
name_list = ["Leo", "Nora", "Emma", "James", "Lucy"]
```

```
name_list[1] = "Clara"  
name_list
```

```
['Leo', 'Clara', 'Emma', 'James', 'Lucy']
```

- Slice assignment

```
name_list[2:4] = ["Ella", "Jasper"]  
name_list
```

```
['Leo', 'Clara', 'Ella', 'Jasper', 'Lucy']
```

List manipulation (2) method: append, insert, remove

- Like strings, Python list objects also support type-specific method calls.

```
name_list = ["Leo", "Nora", "Emma", "James", "Lucy"]
```

```
name_list.append("Henry") # add a new item to the end of a list  
name_list
```

```
['Leo', 'Nora', 'Emma', 'James', 'Lucy', 'Henry']
```

```
name_list.insert(2, 'Mia') # insert an item at given position  
name_list
```

```
['Leo', 'Nora', 'Mia', 'Emma', 'James', 'Lucy', 'Henry']
```

```
name_list.remove("Emma") # remove an item  
name_list
```

```
['Leo', 'Nora', 'Mia', 'James', 'Lucy', 'Henry']
```

List manipulation (2) method: extend

- To append multiple items at the end of a list, you can use `extend`.

```
name_list = ["Leo", "Nora", "Emma", "James", "Lucy"]
```

```
added_name = ['Henry', 'Mia']  
name_list.extend(added_name)  
print(name_list)
```

```
['Leo', 'Nora', 'Emma', 'James', 'Lucy', 'Henry', 'Mia']
```

List manipulation (2) method: sort

- The `sort()` method sorts the elements of a given list in a specific ascending or descending order.

```
number_list = [4, 3, 5, 0, 2, 1]
number_list.sort()
print(number_list)
```

```
[0, 1, 2, 3, 4, 5]
```

- When sorting a list of strings, the strings are rearranged in lexicographic order.

```
name_list = ["Leo", "Nora", "Emma", "James", "Lucy"]
name_list.sort()
print(name_list)
```

```
['Emma', 'James', 'Leo', 'Lucy', 'Nora']
```

Exercise

(B.1) Considering the following lists. Add a new item 'F' to the end of a list.

```
str_list = ['A', 'B', 'C', 'D', 'E']
```

(B.2) Insert an item 'Z' between 'B' and 'C'.

(B.3) Remove 'A' from the list.

(B.4) Sort the items in lexicographic order.

Types of methods

- In-place methods are methods that directly modify the variable they are applied to.
- Out-of-place methods are methods that return a new result without changing the original variable.

Types	Method	In-place methods	Out-of-place method	Return value
List manipulation	append	V		
	insert	V		
	remove	V		
	extend	V		
	pop	V		V
Element search and count	count		V	V
	index		V	V
Sorting and reversing	sort	V		
	reverse	V		

List manipulation (2) method: count

- The `count()` method returns the number of times the specified element appears in the list.

```
item_list = ["apple", "milk", "egg", "apple", "fish", "beef"]
```

```
item_list.count("apple")
```

2

```
item_list.count("beef")
```

1

- Assign the return value to a new variable for future use.

```
cnt = item_list.count("apple")
```

```
print("Number of apples:", cnt)
```

Number of apples: 2

List manipulation (2) method: pop

- There are several ways to delete elements from a list.
 - If you know the value of the element, you can use `remove`.
 - If you know the index of the element, you can use `pop` or `del`.

Case1:

Delete the element at the specified position.

```
mylist = ['a', 'b', 'c', 'd', 'e']  
mylist.pop(1)
```

'b'

The "pop" method returns the removed element.

```
print(mylist)
```

```
['a', 'c', 'd', 'e']
```

Case2:

Delete the last element

```
mylist = ['a', 'b', 'c', 'd', 'e']  
mylist.pop()
```

'e'

```
print(mylist)
```

```
['a', 'b', 'c', 'd']
```

Operators

- Operators are symbols that perform operations on variables or values.

Type of operator	Examples
Arithmetic operators	<code>+, -, *, /</code>
Assignment operators	<code>=, +=, -=</code>
Comparison operators	<code>==, !=, <, ></code>
Logical operators	<code>and, or, not</code>
Membership operators	<code>in, not in</code>
Delete operator	<code>del</code>
Identity operators	<code>is, is not</code>
Bitwise operators	<code>&, , ^</code>

List manipulation (3) operator: del

- If you don't need the deleted item, you can use the `del` operator.
 - Delete the element at the specified position

```
mylist = ['a', 'b', 'c', 'd', 'e']  
del mylist[1]  
print(mylist)  
  
['a', 'c', 'd', 'e']
```

- Delete adjacent elements using list slicing

```
mylist = ['a', 'b', 'c', 'd', 'e']  
del mylist[1:3]  
print(mylist)  
  
['a', 'd', 'e']
```

List manipulation (3) operator: in

- We can use `in` operator to check if an item exists in the list.

```
item_list = ["apple", "milk", "egg", "apple", "fish", "beef"]  
"apple" in item_list
```

True

```
item_list = ["apple", "milk", "egg", "apple", "fish", "beef"]  
"Apple" in item_list
```

False

List manipulation (3) operator: + *

- + operator means concatenation

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
print(c)
```

[1, 2, 3, 4, 5, 6]

- * operator means repetition

```
a*3
```

[1, 2, 3, 1, 2, 3, 1, 2, 3]

List manipulation (4) built-in functions

- There are some built-in functions that can be used on lists that allow you to quickly look through a list.

```
number_list = [3, 41, 12, 9, 74, 15]
```

```
len(number_list) # Number of elements
```

6

```
max(number_list) # Maximum value in the list
```

74

```
min(number_list) # Minimum value in the list
```

3

```
sum(number_list) # The sum of all elements
```

154

Types of functions

Type	Definition	Examples
Built-in Functions	Built-in functions are pre-defined functions that are provided by the programming language itself.	<ul style="list-style-type: none">• <code>print()</code>• <code>len()</code>• <code>type()</code>
Methods	Methods are functions that are specifically defined within the context of certain <u>data types</u> .	<ul style="list-style-type: none">• String methods:<ul style="list-style-type: none">▪ <code>upper()</code>, <code>replace()</code>• List methods:<ul style="list-style-type: none">▪ <code>append()</code>, <code>remove()</code>
User-defined function	A user-defined function is a function that you create yourself to perform a specific task.	<pre>def func(x, y): print (x + y)</pre>

Exercise

(C.1) Considering the following list. Use `pop()` method to remove `'Sun'`. Print the updated list and the deleted element.

Expected result:

```
['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']
```

```
'Sun'
```

```
day_list = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
```

(C.2) Use `del` operator to remove the first two elements in `day_list`.

(C.3) Considering the following list. Check if `'z'` in the list.

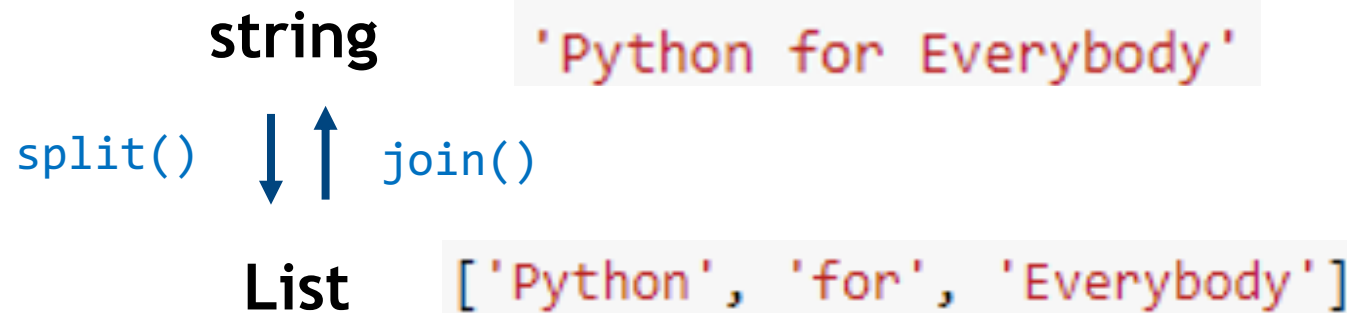
```
mylist = ['x', 'w', 'y', 'x', 'y', 'z', 'y', 'x', 'x', 'w', 'y', 'z']
```

(C.4) Use `mylist` defined in (C.3). Get the number of elements in `mylist`.

(C.5) Use `mylist` defined in (C.3). Count the number of times the value `'x'` appears in the the list.

Lists and strings

- A string is a sequence of characters.
- A list is a sequence of values.
- To convert a variable from a string to a list, you can use `split()`.
- To convert a variable from a list to a string, you can use `join()`.



Lists and strings - split

- Use the `split` method to break a string into words.

```
mystr = 'Python for Everybody'
mystr.split()

['Python', 'for', 'Everybody']
```

- Use an optional argument called a `separator` to specify the word boundaries.

```
mystr = 'Python-for-Everybody'
mystr.split('-')      # mystr.split(sep = '-')

['Python', 'for', 'Everybody']
```

Lists and strings - join

- Concatenates the elements by `join` method
 - `join` is the inverse of `split`.
 - `join` is a string method.

```
mylist = ['Python', 'for', 'Everybody']  
delimiter1 = ' '  
delimiter1.join(mylist)  
  
'Python for Everybody'
```

```
delimiter2 = '-'  
delimiter2.join(mylist)  
  
'Python-for-Everybody'
```

Lists and strings - mutable and immutable

```
# Lists are mutable  
list_1 = [1,2,3]  
list_1.append(4)  
print(list_1)
```

[1, 2, 3, 4]

```
# strings are immutable  
str_1 = "ABC"  
str_1.replace("A","Z")
```

'ZBC'

The "replace" method does not change the contents of the original variable "str1".

str_1

'ABC'

```
str_2 = str_1.replace("A","Z")  
str_2
```

'ZBC'

Exercise

Exercise.D

(D.1) Concatenate the following strings into a string named `mystr` .

Hint: Use math operator.

Expected output:

```
#WWDC21#iphon13#apple#iOS15#MacBook#swift#ios15#swiftui#xcode#apple#MobileAppDevelopment#wwdc21#iosdev"#Apple#AppleEvent#Developer#WWDC"
```

```
tweet1 = "#WWDC21#iphon13#apple#iOS15#MacBook"
tweet2 = "#swift#ios15#swiftui#xcode#apple#MobileAppDevelopment#wwdc21#iosdev"
tweet3 = "#Apple#AppleEvent#Developer#WWDC"
```

(D.2) Convert `mystr` to all uppercase and store the value in a new variable named `mystr_upper` .

(D.3) Convert `mystr_upper` to a list of words.

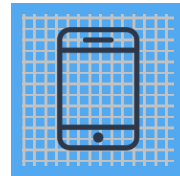
Hint: Use "#" as separator

(D.4) Count the number of times the value 'WWDC21' appears in the list.

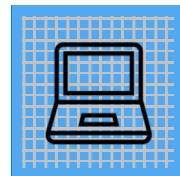
Class and object

- Python is an object-oriented programming (OOP) language. Object-oriented programming is a programming paradigm based on the concept of objects.
- Class: A class is a **blueprint** to create objects.
- Object: An object is an **instance** of a class.

Class - phone



Class - laptop



Object



Class and object

- Example

```
mystr = "hello world"  
print(type(mystr))
```

Create a new object of class “str”.

```
<class 'str'>
```

```
mystr.upper()
```

Functions bound to objects are called methods.

```
'HELLO WORLD'
```

```
mylist = [3, 2, 1]  
print(type(mylist))
```

Create a new object of class “list”.

```
<class 'list'>
```

```
mylist.sort()  
mylist
```

```
[1, 2, 3]
```

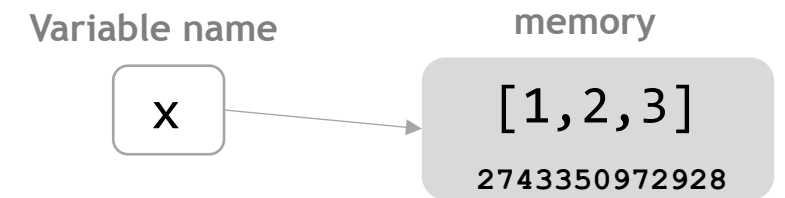

Python variables and memory allocation

- A python variable is a symbolic name, which is a reference to an object. After creating an object, you can refer to it by variable name.
- Use `id()` to see the memory address (object's identity).

```
x = [1,2,3]
```

```
id(x)
```

2743350972928 memory address



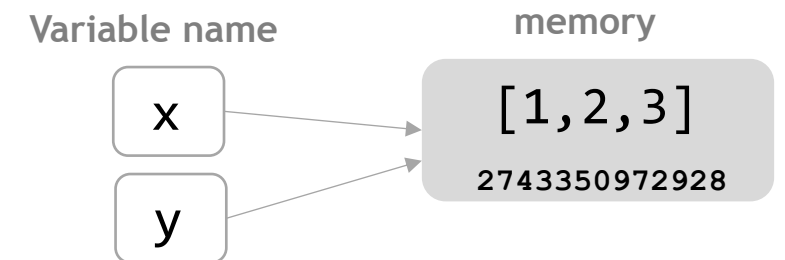
Create an object in memory and let x be a reference to this object.

- If one variable is assigned to another variable, both variables point to the same memory address.

```
y = x
```

```
id(y)
```

2743350972928

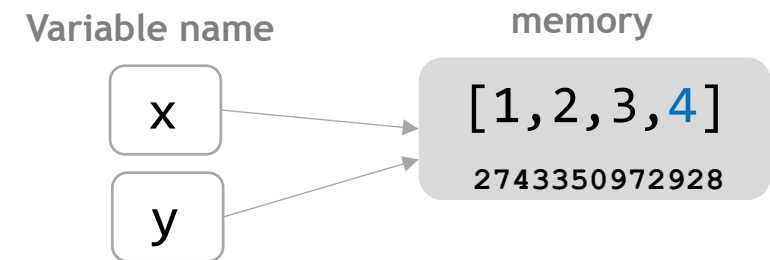


Python variables and memory allocation

- If two variables point to the same address, changing the value of one variable will change the value of the other variable.

```
y.append(4)
print(x)
print(y)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4]
```



- Use the function `copy()` to create a copied variable, and then you can change the copied variable without changing the original variable.

```
z = x.copy()
id(z)
```

```
2743351359168
```

```
z.append(5)
print(x, y, z)
```

```
[1, 2, 3, 4] [1, 2, 3, 4] [1, 2, 3, 4, 5]
```

