FOP

Lists

In this class ...

- How to “save” multiple values to one variable
- What is a list?
- How to access/modify elements on a list?
- How to remove elements from a list?
- How to check if an object is in a list?
- What messages can we send a list?
- How to sort a list?

Motivation

- Say we need to keep track of the number of students in campus for each day of the year
- How many variables would we need?
  - ?
- What a big mess!
- In order to deal with this kind of situation, Python provides a type called `list` that can keep track of many values in one variable

`list`

- A data type that allows storage of objects (i.e. create a collection)
- Lists can contain zero or more objects
- Lists are mutable, i.e. objects can be added and removed from them
  - The objects from classes `int`, `float`, `str`, `bool`, cannot be modified
- Syntax: `my_list = [object1, object2, ...]`
- Empty list: `my_list = []`

WESTMINSTER
International University in Tashkent

Examples

```
>>> grades = [12, 15, 8, 20, 17]
>>> type(grades)
<class 'list'>
>>> grades
[12, 15, 8, 20, 17]
>>> stuff = ['António', 41, 1.78, True]
>>> stuff
['António', 41, 1.78, True]
```

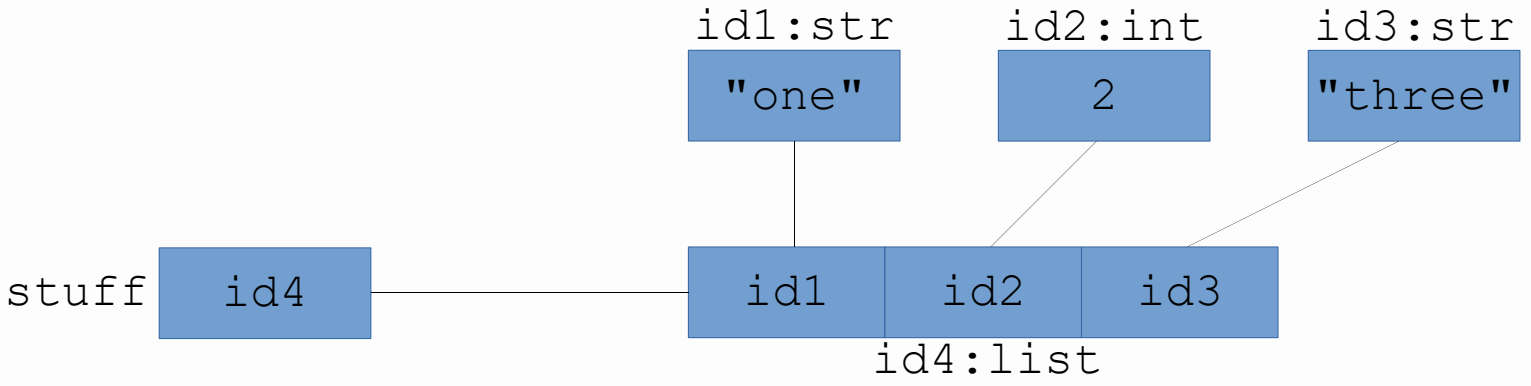
Avoid lists with objects of different types. A list where all objects are of the same type is less prone to errors.

What happens

- In the shell

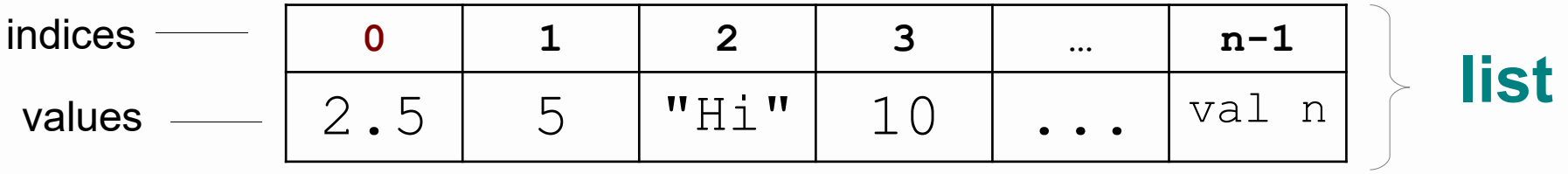
```
>>> stuff = ['one', 2, 'three']
```

- In the memory



Indexing

- To make it simpler, let's pretend that lists keep values and not object references
- The elements in a list are ordered
- The first is at index 0, the second at index 1, and so on (like in C)



- Since the first object is at index `0`, the last element will be at position `n-1`

Accessing elements

- Put the **integer** index of the object we want in brackets after the variable that references the list
- Example:

```
>>> stuff = ['one', 2, 'three']
 >>> stuff[0]
 'one'
 >>> stuff[1]
 2
 >>> stuff[2]
 'three'
 >>> stuff[3]
 Traceback (most recent call last):
```

Getting objects from a list

- As any other objects, the objects in a list can be assigned to a variable
- Example:

```
>>> stuff = ['António', 41, 1.78]
>>> name = stuff[0]
>>> age = stuff[1]
>>> height = stuff[2]
```

- What would be the value of variable `var`?

```
>>> var = stuff[stuff[1]-40]
```

Empty List

- A list with no items

```
>>> my_list = []
```

- What am I trying to do here?

```
>>> my_list[0]
```

- Would there be any problem?

```
Traceback (most recent call last):
   File "<pyshell#17>", line 1, in
 <module>
     my_list[0]
IndexError: list index out of
 range
```

Modifying Lists

- Suppose that the second grade, in the following list, should be 16, instead of 15:

```
>>> grades = [12, 15, 10]
```

- How to fix it?

```
>>> grades[1] = 16
 >>> grades
 [12, 16, 10]
```

- Lists are mutable

Something weird

- Python allows negative indices

	0	1	2	3	4	5
Values	2.5	5	"Hi"	10	6	12
Negative indices	-6	-5	-4	-3	-2	-1

- Both are the same:

```
>>> list[0]
 2.5
 >>> list[-6]
 2.5
```

- This is very handy, since we'll always know where the last item is. Where?

1 minute exercise

- List two different ways to access the last element of any list
- Solution:

```
>>> item = my_list[len(my_list) - 1]
>>> item = my_list[-1]
```

Built-in Functions for Lists (and for other "things" too)

- The following functions **do not modify** the list:

Function	Description
`len(a_list)`	Returns how many objects are on the list (length)
`max(a_list)`	Returns the object in the list with greater value (maximum)*
`min(a_list)`	Returns the object in the list with lower value (minimum)*
`sum(a_list)`	Returns the sum of the numbers in the list
`sorted(a_list)`	Returns a copy of `a_list` sorted in ascending order*

Where have we used `len` , `max`, and `min`, already?

*Items must be comparable

List Concatenation

- Operator + can be used to concatenate lists

```
>>> ['a', 'b', 'c'] + ['d', 'e']
 ['a', 'b', 'c', 'd', 'e']
```

Can only concatenate `list` with `list`.

- Warning!

```
>>> list_1 = ['a', 'b', 'c']
 >>> list_2 = list_1 + ['d', 'e']
 >>> print(list_2)
 ['a', 'b', 'c', 'd', 'e']
 >>> id(list_1[0])
 140484130556648
 >>> id(list_2[0])
 140484130556648
```

Both lists share elements! Not serious for immutable elements.

List Multiplication

```
>>> list_1 = ['a', 'b', 'c']
 >>> list_2 = list_1 * 2
 >>> print(list_2)
 ['a', 'b', 'c', 'a', 'b', 'c']
 >>> id(list_1[0])
 140484130556648
 >>> id(list_2[0])
 140484130556648
 >>> id(list_2[3])
 140484130556648
```

Are you `in`?

- Operator `in` allows to ask if a list contains a given object
- For example:

```
>>> 'a' in ['a', 'b', 'c']
 True
```

Useful with the `if` statement. We've used it with strings before.

- What about this?

```
>>> '2' in ['a', 'hello', 2]
 ?

>>> 2 in ['a', 'b', 2]
 ?
```

Warning!

- Operator `in` verifies if ONE item is in a list
- What is the following program doing?

```
>>> ['a', 'b'] in ['a', 'b', 'c']
 ?
```

- It is verifying if item `['a', 'b']` (which is a list) is in the list `['a', 'b', 'c']`
- There is no list `['a', 'b']` in the list `['a', 'b', 'c']`

List Methods

```
>>> dir(list)
[ …
'append', 'clear', 'copy', 'count',
'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
```

List Methods (...)

Method	Description
`L.append(object)`	Appends the object to list `L`
`L.clear()`	Removes all items from `L`
`L.count(object)`	Returns how many times object is in `L`
`L.extend(iterable)`	Appends the items in the iterable, to `L`
`L.index(object)`	Returns the index of the first occurrence of `object` in `L` (other parameters can be defined)
`L.insert(index, object)`	Inserts object in `L` at a given `index`
`L.pop(`*`index`*`)`	Removes and returns the item at position `index` of `L`. If no `index` is provided, removes and returns the last.
`L.remove(object)`	Removes the first occurrence of object in `L`
`L.reverse()`	Reverses the list (inverts the order, in place)
`L.sort()`	Sorts the list in ascending order (parameter `reverse` can be set to `True`, to sort in decreasing order)

WARNING: all methods in bold **do modify** the original list.

append *vs* extend

```
>>> help(list.append)
Help on method_descriptor:
append(…)
    L.append(object) -> None -- append
object to end

>>> help(list.extend)
Help on method_descriptor:
extend(…)
    L.extend(iterable) -> None -- extend
list by appending elements from the
iterable
```

```
>>> list_1 = ['a', 'b']
>>> list_2 = ['c', 'd']
>>> list_1.append(list_2)
>>> print(list_1)
['a', 'b', ['c', 'd']]
>>> list_1.remove(list_2)
>>> print(list_1)
['a', 'b']
>>> list_1.extend(list_2)
>>> print(list_1)
['a', 'b', 'c', 'd']
```

1 minute problem

- List two ways to sort a list in decreasing order
- Solution 1 (not so efficient):

```
>>> list_1 = [3, 4, 1, 5, 6, 3]
 >>> list_1.sort()
 >>> list_1.reverse()
 >>> print(list_1)
 [6, 5, 4, 3, 3, 1]
```

- Solution 2 (recommended):

```
>>> list_1 = [3, 4, 1, 5, 6, 3]
 >>> list_1.sort(reverse=True)
 >>> print(list_1)
 [6, 5, 4, 3, 3, 1]
```

To use only when the items are comparable to each other.

WESTMINSTER International University in Tashkent

`list.sort()` *vs* built-in `sorted()`

- Would it work with built-in function `sorted`?
  - Check the help on it
- What is the difference between method `sort` and function `sorted`?

REMEMBER!!!

- Like functions, there are methods that don't return any value (in reality, they return `None`)
- What would happen?

```
>>> list_1 = [8, 5, 3, 4]
 >>> list_1 = list_1.sort()
 >>> print(list_1)

?
```

Lists of lists

```
>>> ct = ['Comp Think', 18]
>>> maths = ['Maths', 17]
>>> grades = [ct, maths, ['Intro to Prog',
19]]
>>> print(grades)
[['Comp Think', 18], ['Maths', 17],
['Intro to Prog', 19]]

>>> print(grades[1])
['Maths', 17]

>>> ip = grades[-1]
>>> print(ip)
['Intro to Prog', 19]

>>> print(ip[0])  >>> print(grades[-1][0])
Intro to Prog          Intro to Prog
```

Operator `del`

- Deletes objects
- Example:

```
a = 10
>>> print(a)
10

>>> del a
>>> print(a)
Traceback (most
recent call last):
  File
"<pyshell#127>",
line 1, in <module>
    print(a)
NameError: name 'a'
is not defined
```

- More examples:

```
>>> a_list = ['x',
'y', 'z']
>>> print(a_list)
['x', 'y', 'z']

>>> del a_list
>>> print(a_list)
Traceback (most
recent call last):
  File
"<pyshell#131>",
line 1, in <module>
    print(a_list)
NameError: name
'a_list' is not
```

Deleting inside lists

- We can use `del` to delete objects inside lists
- Just need to provide the location of the object
- Example:

```
>>> a_list = ['x', 'y', 'z']
 >>> del a_list[1]
 >>> print(a_list)
 ['x', 'z']
```

- `del` can be used when we know the index of the object, and `L.remove(object)` can be used when we have a reference to the object, or the object itsef (e.g. `a_list.remove('y')`)

1 minute problem

- Given the following list definition:

```
grades = [['Comp Think', 18], ['Maths', 17],
['Intro to Prog', 19]]
```

how would you:

- Print the grade of Maths?
- >>> ?
- Print the name of the last course?
- >>> ?
- Print the average of the grades?
- >>> ?
- Print the highest grade?
- >>> ?

Solutions

- Given the following list definition:

```
grades = [['Comp Think', 18], ['Maths', 17],
['Intro to Prog', 19]]
```

how would you:

- Print the grade of Maths?
- `>>> print(grades[1][1])`
- Print the name of the last course?
- `>>> print(grades[-1][0])`
- Print the average of the grades?
- `>>> print((grades[0][1]+grades[1][1]+grades[2][1])/3)`

WESTMINSTER
International University in Tashkent

Would this have worked?

```
sum([grades[0][1], grades[1][1],
grades[2][1]])/len(grades)
```

- Print the highest grade?

```
->>> max(grades[0][1], grades[1][1],
grades[2][1])
```

Another 1 minute problem

- `L.index(object)` returns the position of the first occurrence of `object` in list `L`
- If `object` is not present in `L`, it will result in error
- How would you fix the following code to deal with it?

```
a_list = ['x', 'y', 'z']
 obj = input('Object you want to
 find: ')
 position = a_list.index(obj)
 print(obj, 'is in position',
 position)
```

Try running it as it is and input something that is not in the list.

Solution

```
a_list = ['x', 'y', 'z']
obj = input('You want the position of? ')
if obj in a_list:
    position = a_list.index(obj)
    print(obj, 'is in position', position)
else:
    print(obj, 'is not present in the list')
```

Just a curiosity …

- Strings can also be indexed in the same way we do lists
- Example:

```
university = 'Aberystwyth'
 print(university[0])
 print(university[-1])
```

Back to the questions

- How to “save” multiple values to one variable
- What is a list?
- How to access/modify elements on a list?
- How to remove elements from a list?
- How to check if an object is in a list?
- What messages can we send a list?
- How to sort a list?

Miscellaneous

- Interesting topic to explore on your own
  - List slicing Read about it and do the exercises in the seminars

Further reading

- PP, chapter 8