COMP1730/COMP6730
Programming for Scientists

More about lists

Lecture outline

- **Lists**
- Mutable objects & references

Sequence data types (recap)

- A *sequence* contains $n \geq 0$ values (its *length*), each at an *index* from 0 to $n - 1$.
- python's built-in sequence types:
  - strings (`str`) contain only characters;
  - lists (`list`) can contain a mix of value types;
  - tuples (`tuple`) are like lists, but immutable.
- Sequence types provided by other modules:
  - e.g., NumPy arrays (`numpy.ndarray`).

Lists

- python's `list` is a general sequence type: elements in a `list` can be values of any type.
- List literals are written in square brackets with comma-separated elements:

```
>>> a_list_of_ints = [2, -4, 2, -8 ]
>>> a_date = [28, "August", 2023]
>>> type(a_date)
<class 'list'>

>>> list("abcd")
['a', 'b', 'c', 'd']

>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

List comprehension

Create a list by evaluating an expression for each element in a sequence or *iterable* data (later in the course):

```
output_list = [ expression for item in iterable ]
```

This is equivalent to:

```
output_list = []
for item in iterable:
    output_list.append(expression)
```

Example:

```
>>> [ord(c) for c in "abcd"]
[97, 98, 99, 100]

>>> [ 1/x for x in range(1,6) ]
[1.0, 0.5, 0.3333333, 0.25, 0.2]
```

Australian National University

Conditional list comprehension

Conditional list comprehension selects only elements that satisfy a condition:

```
output_list = [ expression for item in iterable if condition ]
```

This is equivalent to:

```
output_list = []
for item in iterable:
    if condition:
        output_list.append(expression)
```

Example:

```
>>> [ i for i in range(2,12) if 12 % i == 0 ]
[2, 3, 4, 6]
```

Australian National University

Lists of lists

Elements of a list can be a list:

```
>>> A = [ [1, 2], [3, 4, 5], [6, 7, 8, 9] ]
>>> A[0]
[1, 2]
>>> A[1][2]
5
>>> A[0:1]
[ [1, 2] ]
>>> A[0:1][1:]
[ ]
>>> A[0:1][1]
IndexError: list index out of range
```

- Indexing a list returns an element, but slicing a list returns a list.
- Indexing and slicing associate to the left: `a_list[i][j] == (a_list[i])[j]`.

Australian National University

Operations on lists

- Use '+' operator to concatenate lists:

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

- Use '*' operator of a `list` and an `int` to repeat a list:

```
>>> 3 * [1, 2]
[1, 2, 1, 2, 1, 2]
>>> [1, 2] * 3
[1, 2, 1, 2, 1, 2]
```

- Equality, *`list == list`*, and ordering comparisons, *`list < list`*, *`list >= list`*, etc, work the same way as for other (standard) sequence types, such as strings.

Lecture outline

* Lists
* **Mutable objects & references**

Australian National University

Values are objects

* In python, every value is an *object*.
* Every object has a unique(*) identifier.

```
>>> id(1)
136608064
```

(Essentially, its location in memory.)

* *Immutable* objects never change.
  - For example, numbers (`int` and `float`), strings and tuples.
* *Mutable* objects can change.
  - For example, lists.

Australian National University

Immutable objects

* Operations on immutable objects create new objects, leaving the original unchanged.

```
>>> a_string = "spam"
>>> id(a_string)
3023147264
>>> b_string = a_string.replace('p', 'l')
>>> b_string
'slam'
>>> id(b_string)
3022616448
>>> a_string
'spam'
```

not the same!

Australian National University

Mutable objects

* A mutable object can be modified yet it's identity remains the same.
* Lists can be modified through:
  - element and slice assignment; and
  - modifying methods/functions.

Element & slice assignment

```
>>> a_list = [1, 2, 3]
>>> id(a_list)
3022622348 ←
>>> b_list = a_list
>>> a_list[2] = 0
>>> b_list
[1, 2, 0]
>>> b_list[0:2] = ['A', 'B']
>>> a_list
['A', 'B', 0]
>>> id(b_list)
3022622348 ←
```

the same object!

Australian National University

Modifying list methods

* *a_list*.append(*new element*)
* *a_list*.insert(*index*, *new element*)
* *a_list*.pop(*index*)
  - *index* defaults to -1 (last element).
* *a_list*.remove(*a value*)
* *a_list*.extend(*an iterable*)
* *a_list*.sort()
* *a_list*.reverse()
* Note: Most do not return a value.

Australian National University

Lists contain references

* Assignment associates a (variable) name with a *reference* to a value (object).
  - The variable still references the same object (unless reassigned) even if the object is modified.
* *A list contains references to its elements.*
* Slicing a list creates a new list, but containing references to the same objects ("shallow copy").
* Slice assignment *does not copy*.

Australian National University

```
a_list = [1,2,3]
b_list = a_list
a_list.append(4)
print(b_list)
```

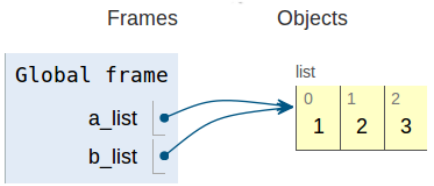

Image from pythontutor.com

```
a_list = [1,2,3]
b_list = a_list[:]
a_list.append(4)
print(b_list)
```

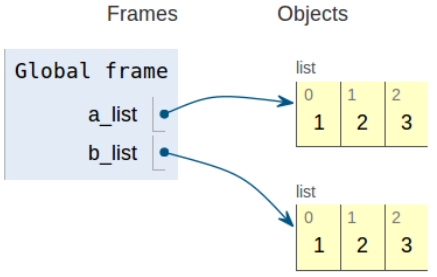

Image from pythontutor.com

```
a_list = [ [1,2], [3,4] ]
b_list = a_list[:]
a_list[0].reverse()
b_list.reverse()
print(b_list)
```

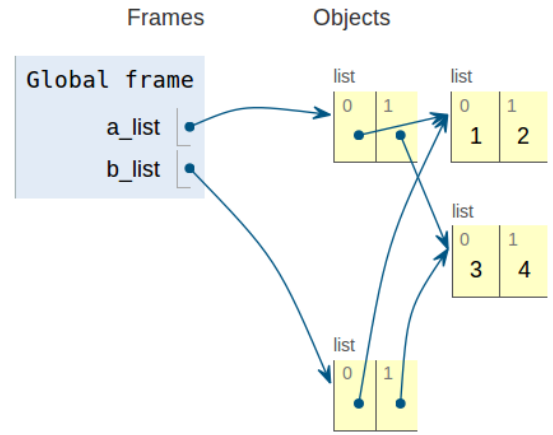

Image from pythontutor.com

Australian National University

```
a_list = [ [1,2], [3,4] ]
b_list = a_list[:]
a_list[0] = a_list[0][::-1]
b_list.reverse()
print(b_list)
```

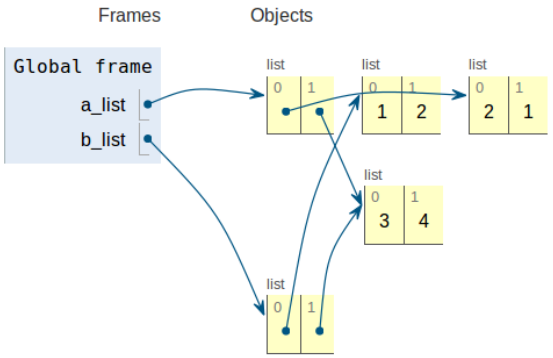

Image from pythontutor.com

Australian National University

```
a_list = [1,2,3]
b_list = [4,5,6]
a_list.append(b_list)
c_list = a_list[:]
b_list[0] = 'A'
```

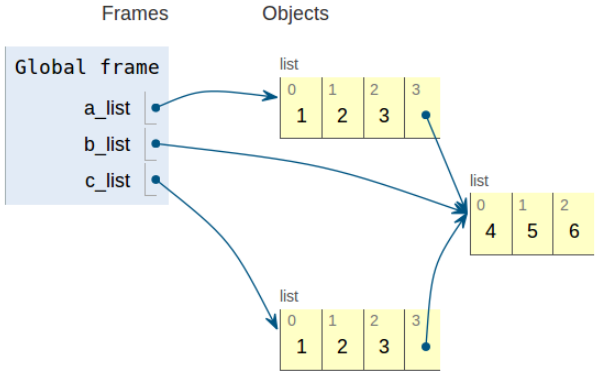

Image from pythontutor.com

Australian National University

Can you find the mistakes below?

```
# example 1
a_list = [3,1,2]
a_list = a_list.sort()

# example 2
a_list = [1,2,3]
b_list = a_list
a_list.append(b_list)

# example 3
a_list = [[]] * 3
a_list[0].append(1)
```

(added after lecture)
Polling for: What is the value of `a_list` after?

```
a_list = [3,1,2]
a_list = a_list.sort()
```

* [1, 2, 3]: 43 votes
* [3, 2, 1]: 2 votes
* None: 8 votes (correct answer)

```
a_list = [[]] * 3
a_list[0].append(1)
```

* [[1], [1], [1]]: 12 votes (correct answer)
* [[1], [], []]: 22 votes
* [1, [], []]: 16 votes
* I don't know: 2 votes