ECOR 1042
Data Management

Lists, Tuples and Sets

Carleton University

References

- *Practical Programming*, 3rd ed.
  - Chapter 8, *Storing Collections of Data Using Lists*
  - Chapter 11, *Storing Data Using Other Collection Types*
    - *Storing Data Using Sets* (pp. 203 - 209)
    - *Storing Data Using Tuples* (pp. 209-214)

Lecture Objectives

- Review Python's `list` type
- More on lists: Slices, Aliasing, List methods, Nested lists, Functions that modify their list arguments
- Introduce Python's `tuple` and `set` types

Learning Outcomes (Vocabulary)

- Know the meaning of these words and phrases
  - List, tuple, set (types `list`, `tuple`, `set`)
  - Ordered collection, unordered collection
  - Mutable collection, immutable collection

Learning Outcomes

- Be able to evaluate expressions consisting of `list`, `tuple` and `set` objects and some of the operations supported by those types
- Understand the key differences between lists, tuples and sets

In ECOR1041 you learned/used:

- **Simple Types** – float, int, Boolean
- **Aggregate Types** (Collections)

	Strings	Lists	Sets	Tuples
Ordered	Yes	Yes	No	Yes
Mutable	No	Yes	Yes	No
Duplicates allowed	Yes	Yes	No	Yes
Notation	" "	[ ]	{ }	( )

Lists

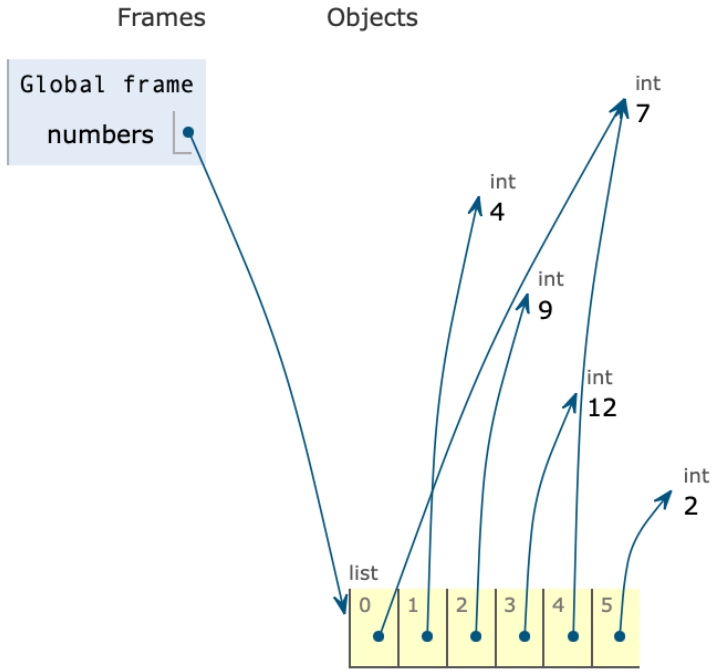
Review: Lists

- In computer science, a list is a finite sequence of values; e.g., 7, 4, 9, 12, 7, 2 is a list of six integers
- Python: create a new list (an object of type `list`)

```
>>> numbers = [7, 4, 9, 12, 7, 2]
>>> numbers
[7, 4, 9, 12, 7, 2]
```

Review: Type `list`

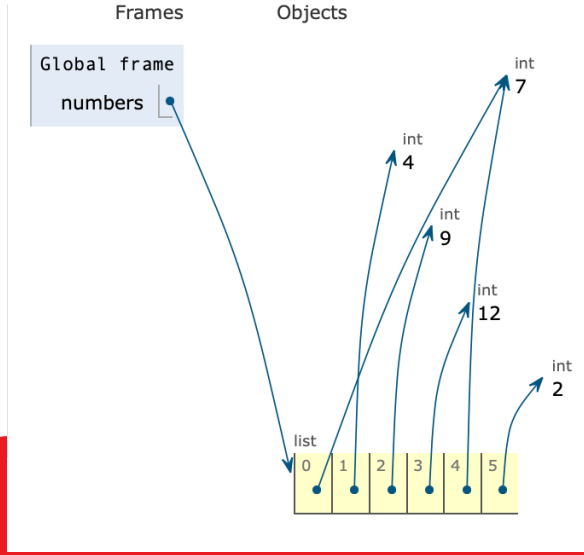
- A list stores references to objects (memory addresses of objects)
  - `numbers` refers to a `list` object that stores six references to objects of type `int`



Review: A `list` is an Ordered Collection

- An integer *index* specifies the location of each list element
  - In a list with *n* elements, indices range from -*n* to *n* - 1

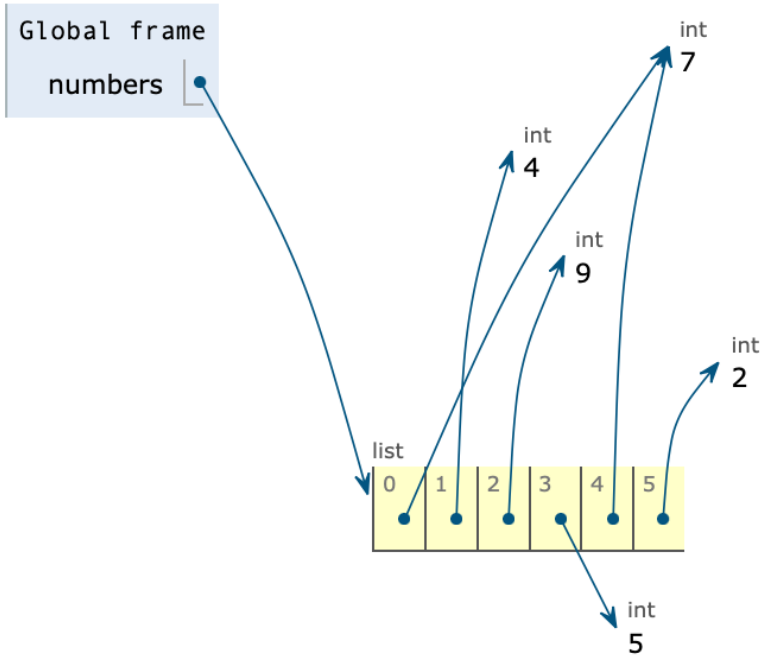
```
>>> numbers[0]
7
>>> numbers[5]
2
>>> numbers[-1]
2
>>> numbers[-6]
7
```



Review: A `list` is a Mutable Collection

- An assignment operation can be used to replace the list element at a specified location
- Example: replace the 4th element with 5

```
>>> numbers[3] = 5
>>> numbers
[7, 4, 9, 5, 7, 2]
```



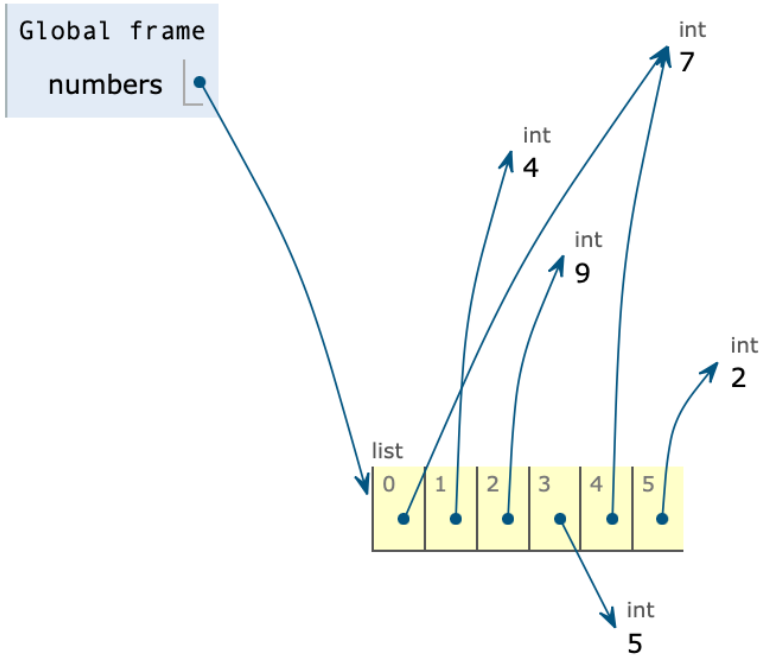
Slicing Lists

- Create a **new** list with consecutive elements of the original list from index i (included) to index j (excluded)

```
new_list = original_list[i:j]
```

  - `i` omitted – from the beginning of the list
  - `j` omitted – until the end of the list

```
>>> numbers[:]
[7, 4, 9, 12, 7, 2]
>>> numbers[2:4]
[9, 12]
```



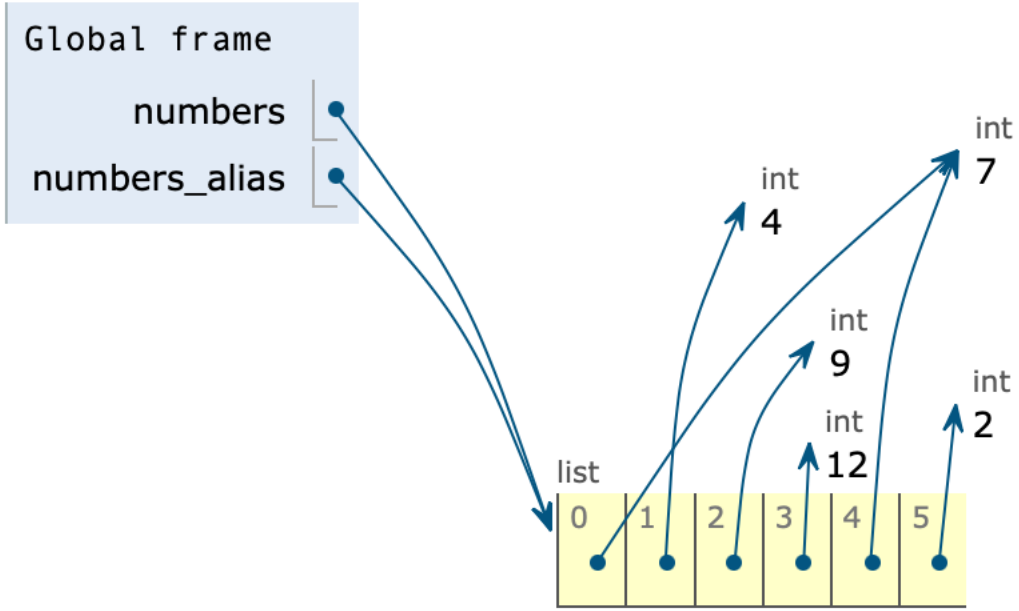
Aliasing

- Two variables are aliases if they refer to the same object (contain the same memory address)

```
>>> numbers = [7, 4, 9, 12, 7, 2]
>>> numbers_alias = numbers
```

- The = operation initializes `numbers_alias` with a copy of the reference stored in `numbers`
- `numbers` and `numbers_alias` refer to the same list (see next slide)

Aliasing



Aliasing

- Understanding aliasing is important when working with mutable objects
- Changes made to one object (e.g., `numbers`) are "seen" by its aliases (e.g., `numbers_alias`)

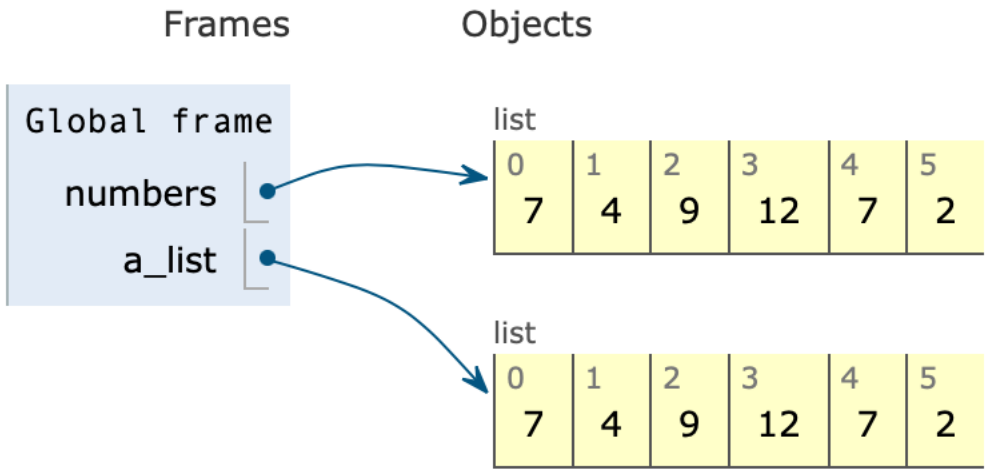
```
>>> numbers[3] = 5
>>> numbers
[7, 4, 9, 5, 7, 2]
>>> numbers_alias
[7, 4, 9, 5, 7, 2]
```

Aliasing

```
>>> numbers = [7, 4, 9, 12, 7, 2]
>>> a_list = [7, 4, 9, 12, 7, 2]
>>> numbers == a_list
True
```

- `a_list = [7, 4, 9, 12, 7, 2]` creates a new list that is identical to the one referred to by `numbers` (see next slide)
- `numbers` and `a_list` are not aliases, even though they refer to identical lists

Aliasing



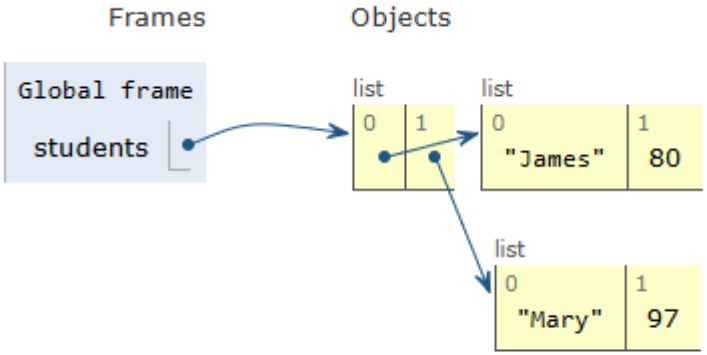
List methods

- Provide useful operations to work with lists
  - L.append(v) - Appends value v to list L.
  - L.clear() - Removes all items from list L.
  - L.count(v) - Returns the number of occurrences of v in list L.
  - L.extend(v) - Appends the items in v to L.
  - L.index(v) - Returns the index of the first occurrence of v in L—an error is raised if v does not occur in L.
  - L.insert(i, v) - Inserts value v at index i in list L, shifting subsequent items to make room
  - L.pop() - Removes and returns the last item of L (which must be nonempty).
  - L.remove(v) - Removes the first occurrence of value v from list L.
  - L.reverse() - Reverses the order of the values in list L.
  - L.sort() - Sorts the values in list L in ascending order (for strings with the same letter case, it sorts in alphabetical order).

Nested Lists

- The elements of a list can be another list

```
>>> students = [["James", 80], ["Mary", 97]]
```

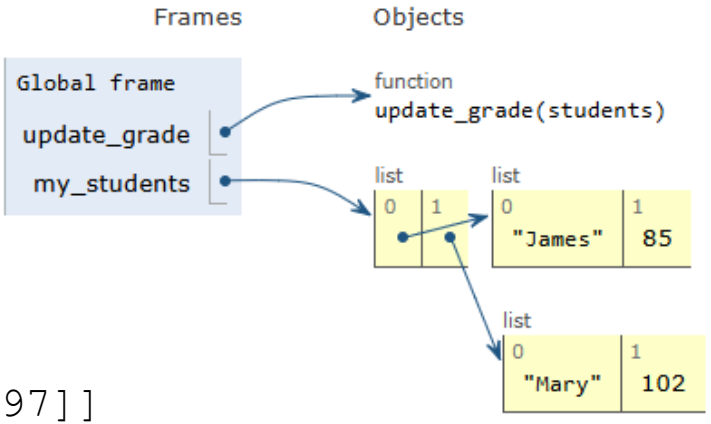


Functions that modify their list arguments

- Lists are passed to functions by reference

```
def update_grade(students):
    for student in students:
        student[1] +=5

my_students = [["James", 80], ["Mary", 97]]
update_grade(my_students)
```



Tuples

Tuples

- Python provides a built-in type named `tuple`
- A `tuple` is a finite sequence of values and is an ordered collection (like a `list`)
- Unlike a `list`, a `tuple` is immutable (cannot be modified after it is been created)

Tuples

- A `tuple` object is created by an expression of the form `(`*`expression1`*`, `*`expression2`*`, ..., `*`expressionN`*`)`
  - Important: expressions enclosed by `()`, not `[]`

```
>>> values = (7, 4, 9, 12, 7, 2)
>>> values
(7, 4, 9, 12, 7, 2)
```

Tuples

- The `()`'s are not required
- The commas in `7, 4, 9, 12, 7, 2` specify that Python should construct a tuple

```
>>> values = 7, 4, 9, 12, 7, 2
>>> values
(7, 4, 9, 12, 7, 2)
```

Tuples

- Any operation supported by type `list` that does not modify a list is also supported by type `tuple`

```
>>> values = (7, 9, 2)
>>> values[1]   # access by index
9
>>> values[-3]  # negative indices are supported
7
>>> 2 in values  # containment testing
True
>>> len(values)
3
```

Tuples

- More tuple operations

```
>>> values + (4, 9, 3)   # concatenation
(7, 9, 2, 4, 9, 3)

>>> values * 2           # replication
(7, 9, 2, 7, 9, 2)

>>> for elem in values:     # iteration
        print(elem)
```

Tuples

- Tuples are immutable collections

```
>>> values = (7, 4, 9, 12, 7, 2)
>>> values[3] = 5
builtins.TypeError: 'tuple' object does not support
item assignment

>>> del values[4]
builtins.TypeError: 'tuple' object does not support
item deletion
```

Why Use Tuples?

- Tuples can be very useful to aggregate data together
- Some data “belongs” together
- **Example:**
  - **Person’s name:** name, surname
  - **Birthday:** day, month, year
  - **Address:** number, street, city, postal code
  - **Location:** latitude, longitude

Tuple Packing and Unpacking

Although tuples are ordered (and thus can be indexed), usually we pack and unpack values

- Pack two values into a tuple

```
>>> point = (10, 20)
```

- Later, unpack the tuple into two variables

```
>>> x, y = point
>>> x
10
>>> y
20
```

Tuple Packing and Unpacking

- Combining packing and unpacking lets us assign objects to multiple variables in a single statement

```
>>> (x, y) = (10, 20)
>>> x, y = 10, 20
```

Tuple Packing and Unpacking

- Tuple packing and unpacking is often used to swap the objects that bound to two variables

```
>>> x = 10
>>> y = 20
>>> y, x = x, y
>>> x
20
>>> y
10
```

The `int` objects bound to `x` and `y` are stored into a tuple, then those objects are assigned to the variables on the left side of `=`

Sets

Sets

- Python provides a built-in type named `set`
- A `set` is a unordered collection of distinct values (duplicate values are not permitted)
- Like a `list`, a `set` is mutable

Sets

- A `set` object is created by an expression of the form
  `{`*`expression1`*`, `*`expression2`*`, ..., `*`expressionN`*`}`

```
>>> s = {7, 12, 4, 7, 9, 18, 9, 2}
>>> s
{2, 18, 4, 7, 9, 12}
```

- Notice that
  - the duplicate 7 and 9 are not stored in the set
  - when `s` is evaluated, the original order of elements is not maintained

Set Operations

```
>>> s = {7, 12, 4, 7, 9, 18, 9, 2}
>>> len(s) # cardinality of s (no. of elems in s)
6
>>> min(s)
2
>>> max(s)
18
>>> 4 in s # test for membership
True
>>> 9 not in s
False
```

Set Operations

- A `set` is an unordered collection, so sequence-like operations that specify a position (index) are not supported

```
>>> s = {7, 12, 4, 7, 9, 18, 9, 2}
>>> s[5]
builtins.TypeError: 'set' object is not subscriptable
>>> s[2] = 14
builtins.TypeError: 'set' object does not support item
assignment
>>> del s[1]
builtins.TypeError: 'set' object does not support item
deletion
```

Set Operations

- Some operations are provided by *methods*

```
>>> s = {7, 12, 4, 7, 9, 18, 9, 2}
>>> s.add(20) # insert 20 in the set
>>> s.add(7)  # duplicate 7 is discarded
>>> s
{2, 18, 4, 20, 7, 9, 12}
>>> s.remove(4)
>>> s
{2, 18, 20, 7, 9, 12}
>>> s.pop()   # remove and return an arbitrary element
2
```

Set Operations

- Type `set` provides operators and methods that perform mathematical operations on sets (union, intersection difference, symmetric difference, is-superset, is-subset)
- These will be introduced throughout the course, as required

Syntax: Caution

- Empty list: `l = []` or `l = list()`
- Empty tuple: `t = ()` or `t = tuple()`
- Empty set: `s = set()  # {}` creates a dictionary
- 1-element list: `l = [42]`
- 1-element set: `l = {42}`
- 1-element tuple: `t = (42,)` or `t = 42,`
  - Without the comma, t will be assigned an `int` (42)

Collections

	Strings	Lists	Sets	Tuples
Ordered	Yes	Yes	No	Yes
Mutable	No	Yes	Yes	No
Duplicates allowed	Yes	Yes	No	Yes
Notation	" "	[ ]	{ }	( )
Declare an Empty	S = ""	L = [ ]	S = set()	T = ()
Declare a 1-element	S = "1"	L = [1]	S = {1}	T = (1,)

- The comma is required for 1-element tuples
- It is known as the tuple constructor

Practice for Home!

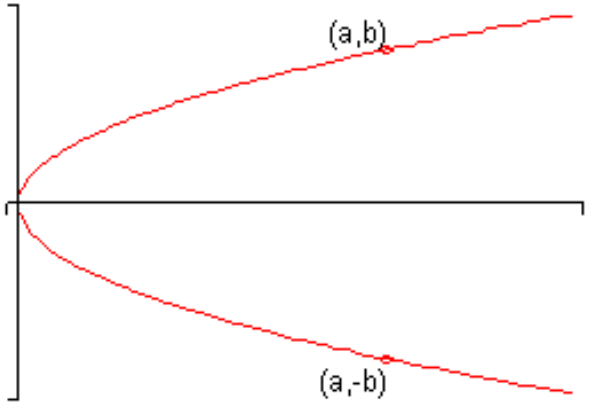
Syntax Exercise

- What will the script print?

```
1  t1 = ("1")
2  print(type(t1))
3  t2 = ("1",)
4  print(type(t2))
5  t3 = ("1","2")
6  print(type(t3))
```

Function that Returns a Tuple

- Write a function to find the symmetric value about the x axis. The function returns a tuple.
  - Version 1: The function takes two input parameters, the x and y coordinates
  - Version 2: The function takes one input parameter, a point



Another Exercise on Tuples

a) **Write a function that returns the roots of a quadratic equation.**

b) **Write a script that tests the function**

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Recap Learning Outcomes

- Review Python's `list` type
- More on lists: Slices, Aliasing, Functions that modify their list arguments, List methods, Nested lists
- Introduce Python's `tuple` and `set` types

ECOR 1042
Data Management

Lists, Tuples and Sets