

# SEQUENCE DATA TYPES

DR IAN CORNELIUS

# HELLO

- Learning Objectives:
  1. Understand the other sequence data types that are built-in to Python
  2. Demonstrate the ability to use these other sequence data types

## PREVIOUSLY...

- Last week you were introduced to a couple of sequence data types:
  - Bytes
  - Range
- This week, you will be introduced to the other two:
  - Lists
  - Tuples

# INTRODUCTION TO LISTS

- Lists are used to store multiple items into a single variable
- They are considered to be:
  - **ordered**: the items have a defined order and this order will not change when new items are added to the list
  - **changeable**: the items of a list are mutable (can be changed), added or removed
  - **allowable of duplicates**: lists are indexed, and therefore items in a list can be duplicated
- The size of a list (or the number of items stored in a list) can be determined using the `len()` function

## CREATING A LIST

- Lists are created by using a set of square brackets (`[]`)
- Other data types can be type-casted as a list by using its constructor

```
</> listExample1 = ["4061CEM", "Programming", "Algorithms"]  
listExample2 = list(("4061CEM", "Programming", "Algorithms"))  
listExample3 = list("Hello 4061CEM!")
```

## ITEMS OF A LIST

- The items of a list can be any data type
  - i.e. it can be a mixture of data types such as booleans, strings, lists or integers

```
</> listExample1 = ["4061CEM", "Programming", "Algorithms"]  
listExample2 = [4061, "Programming and Algorithms", True,  
                ["Dr Ian Cornelius", "Mr Terry Richards"]]
```

## ACCESSING LIST ITEMS (1)

- The items in a list can be accessed by referring to its index number inside a set of square brackets (`[]`)
  - **Note** that the index of a list begins at `0` in Python, other programming languages begin at `1`

```
</> listExample1 = ["4061CEM", "Programming", "Algorithms"]
      listExample2 = [4061, "Programming and Algorithms", True,
                      ["Dr Ian Cornelius", "Mr Terry Richards"]]
```

```
</> listExample1[1]
      listExample1[1].upper()
      listExample2[3][1]
```

```
▶ listExample1[1] = Programming
   listExample1[1].upper() = PROGRAMMING
   listExample2[3][1] = Mr Terry Richards
```

## ACCESSING LIST ITEMS (2)

### NEGATIVE INDEXING

- When using a negative index, it will access the list from the end
  - i.e. `-1` will refer to the last item and `-2` the second to last item etc.

```
</> listExample1 = ["4061CEM", "Programming", "Algorithms"]
```

```
</> listExample1[-1]  
listExample1[-2]
```

```
▶ listExample1[-1] = Algorithms  
listExample1[-2] = Programming
```



## ACCESSING LIST ITEMS (3)

### SLICING A LIST USING POSITIVE INDEXES I

- A selection of items in a list can be returned using a `slice`
  - a slice is a number range using a colon (":") between the two numbers
  - i.e. `1:3` represents begin at index `1` and go up to index `3`
- The search will begin at the start value and include it in the returned list
  - it will end at the end value, but will **not** include it in the returned list

```
</> listExample1 = [4061, "Programming and Algorithms", True, "Dr Ian Cornelius"]
```

```
</> listExample1[1:3]
```

```
▶ listExample1[1:3] = ['Programming and Algorithms', True]
```

## ACCESSING LIST ITEMS (4)

### SLICING A LIST USING POSITIVE INDEXES II

- By not providing a start value, the range function will always begin at the first index

```
</> listExample1 = [4061, "Programming and Algorithms", True, "Dr Ian Cornelius"]
```

```
</> listExample1[:3]
```

```
▶ listExample1[:3] = [4061, 'Programming and Algorithms', True]
```

- If you do not provide an end value, it will return all items from the start index to the end of the list

```
</> listExample1[2:]
```

```
▶ listExample1[2:] = [True, 'Dr Ian Cornelius']
```

## MODIFYING A LIST (1)

- As items are ordered and indexed, they are modifiable; otherwise known as being **mutable**

## INSERTING AN ITEM

- Items can be inserted into the list and not replace pre-existing items at a given index using the `insert()` function

```
</> listExample1 = ["4061CEM", "Programming", "Algorithms"]
```

```
</> listExample1.insert(3, "Dr Ian Cornelius")
```

```
▶ listExample1 = ['4061CEM', 'Programming', 'Algorithms', 'Dr Ian Cornelius']
```

- You can also insert an item at a different index
  - this will move the item at the index to the right by one, and all other items

```
</> listExample1.insert(1, "Dr Ian Cornelius")
```

```
▶ listExample1 = ['4061CEM', 'Dr Ian Cornelius', 'Programming', 'Algorithms']
```

## MODIFYING A LIST (2)

### APPENDING AN ITEM

- Items can be inserted at the end of the list using the `append()` function

```
</> listExample1 = ["4061CEM", "Programming", "Algorithms"]
```

```
</> listExample1.append("Mr Terry Richards")
```

```
▶ listExample1 = ['4061CEM', 'Programming', 'Algorithms', 'Mr Terry Richards']
```

## MODIFYING A LIST (3)

### REMOVING ITEMS FROM A LIST I

- Items can be removed from a list using the `remove()` function
- This will search the list for a specific value and then remove it

```
</> listExample1 = ["4061CEM", "Programming", "Algorithms"]
```

```
</> listExample1.remove("Programming")
```

```
▶ listExample1 = ['4061CEM', 'Algorithms']
```

## MODIFYING A LIST (4)

### REMOVING ITEMS FROM A LIST II

- An item can also be removed from a list by using the `pop()` function
  - this will remove the item from a list by its index

```
</> listExample1 = ["4061CEM", "Programming", "Algorithms"]
```

```
</> listExample1.pop(1)
```

```
▶ listExample1 = ['4061CEM', 'Algorithms']
```

## MODIFYING A LIST (5)

### REMOVING ITEMS FROM A LIST III

- An item can also be removed from a list by referring to its index in square brackets ([]) and using the `del` keyword

```
</> listExample1 = ["4061CEM", "Programming", "Algorithms"]
```

```
</> del listExample1[1]
```

```
▶ listExample1 = ['4061CEM', 'Algorithms']
```



## MODIFYING A LIST (6)

### CLEARING A LIST

- A list can be cleared of all its items, but still reserve its memory location by using the `clear()` function
- This will empty the contents of a list and leave it empty, symbolised by just the square brackets (`[]`)

```
</> listExample1 = ["4061CEM", "Programming", "Algorithms"]
```

```
</> listExample1.clear()
```

```
▶ listExample1 = []
```

## MERGING A LIST (1)

- There are various methods of merging two lists together: concatenation or extension

## CONCATENATING A LIST

- The items of a list can be concatenated with the items of another list using the `+` operator

```
</> listExample1 = ["4061CEM", "Programming and Algorithms 1", "Dr Ian Cornelius"]  
listExample2 = ["4059CEM", "Legal and Ethical Foundations", "Mr Terry Richards"]
```

```
</> mergedListExample1 = listExample1 + listExample2
```

```
▶ mergedListExample1 = ['4061CEM', 'Programming and Algorithms 1', 'Dr Ian Cornelius', '4059CEM', 'Legal and Ethical  
Foundations', 'Mr Terry Richards']
```

## MERGING A LIST (2)

### EXTENDING A LIST

- The items of a list can be merged with the items of another list using the `extend()` function

```
</> listExample1 = ["4061CEM", "Programming and Algorithms 1", "Dr Ian Cornelius"]  
    listExample2 = ["4059CEM", "Legal and Ethical Foundations", "Mr Terry Richards"]
```

```
</> listExample1.extend(listExample2)
```

```
▶ listExample1 = ['4061CEM', 'Programming and Algorithms 1', 'Dr Ian Cornelius', '4059CEM', 'Legal and Ethical  
Foundations', 'Mr Terry Richards']
```

## COPYING A LIST

- The method of copying a list by using `list2 = list1` is incorrect
  - this method creates a reference to `list1` and not an actual copy; therefore any changes made in `list1` will occur in `list2`
- The correct process of copying a list can be achieved by the `copy()` function or the `list()` constructor itself

```
</> listExample1 = ["4061CEM", "Programming", "Algorithms"]
```

```
</> copyListExample1 = listExample1.copy()  
copyListExample2 = list(listExample1)
```

```
▶ copyListExample1 = ['4061CEM', 'Programming', 'Algorithms']  
copyListExample2 = ['4061CEM', 'Programming', 'Algorithms']
```

# INTRODUCTION TO TUPLES

- Tuples are used to store multiple items into a single variable
- They are considered to be:
  - **ordered**: the items have a defined order and this order will not change
  - **unchangeable**: the items of a tuple are immutable; they cannot be changed, added or removed
  - **allowable of duplicates**: tuples are indexed, and therefore items in a list can be duplicated
- The size of a tuples (or the number of items stored in a tuple) can be determined using the `len()` function

## CREATING A TUPLE

- Tuples are created by using a set of brackets `()`
- Other data types can be type-casted as a tuple by using its constructor

```
</> tupleExample1 = ("4061CEM", "Programming", "Algorithms")
tupleExample2 = tuple("Hello 4061CEM")
```

```
▶ tupleExample1 = ('4061CEM', 'Programming', 'Algorithms')
tupleExample2 = ('H', 'e', 'l', 'l', 'o', ' ', '4', '0', '6', '1', 'C', 'E', 'M')
```

## ITEMS OF A TUPLE

- The items of a tuple can be any data type
  - i.e. it can be a mixture of data types such as booleans, strings, tuples or integers

```
</> tupleExample1 = ("4061CEM", "Programming", "Algorithms")
tupleExample2 = (4061, "Programming and Algorithms", True,
                ("Dr Ian Cornelius", "Mr Terry Richards"))
```

```
▶ tupleExample1 = ('4061CEM', 'Programming', 'Algorithms')
tupleExample2 = (4061, 'Programming and Algorithms', True, ('Dr Ian Cornelius', 'Mr Terry Richards'))
```

## ACCESSING TUPLE ITEMS (1)

- The items in a tuple can be accessed by referring to its index number inside a set of square brackets ([])
  - **Note** that the index of a tuple begins at 0 in Python, other programming languages begin at 1

```
</> tupleExample1 = ("4061CEM", "Programming", "Algorithms")
```

```
</> tupleExample1[1]  
tupleExample1[2]
```

```
▶ tupleExample1[1] = Programming  
tupleExample1[2] = Algorithms
```



## ACCESSING TUPLE ITEMS (2)

### NEGATIVE INDEXING

- When using a negative index, it will access the tuple from the end
  - i.e. `-1` will refer to the last item and `-2` the second to last item etc.

```
</> tupleExample1 = ("4061CEM", "Programming", "Algorithms")
```

```
</> tupleExample1[-1]  
tupleExample1[-2]
```

```
▶ tupleExample1[-1] = Algorithms  
tupleExample1[-2] = Programming
```

## ACCESSING TUPLE ITEMS (3)

### SLICING A TUPLE USING POSITIVE INDEXES I

- A selection of items in a tuple can be returned using a `slice`
  - a slice is a number range using a colon (":") between the two numbers
  - i.e. `1:3` represents begin at index `1` and go up to index `3`
- The search will begin at the start value and include it in the returned tuple
  - it will end at the end value, but will **not** include it in the returned tuple

```
</> tupleExample1 = (4061, "Programming and Algorithms", True, "Dr Ian Cornelius")
```

```
</> tupleExample1[1:3]
```

```
▶ tupleExample1[1:3] = ('Programming and Algorithms', True)
```

## ACCESSING TUPLE ITEMS (4)

### SLICING A TUPLE USING POSITIVE INDEXES II

- By not providing a start value, the range function will always begin at the first index

```
</> tupleExample1 = (4061, "Programming and Algorithms", True, "Dr Ian Cornelius")
```

```
</> tupleExample1[:3]
```

```
▶ tupleExample1[:3] = (4061, 'Programming and Algorithms', True)
```

- By not providing an end value, the range function will always terminate at the last index

```
</> tupleExample1[2:]
```

```
▶ tupleExample1[2:] = (True, 'Dr Ian Cornelius')
```

## MODIFYING A TUPLE (1)

- The items of a tuple cannot be changed once they have been created, commonly referred to as **immutable**
- There is a workaround to changing the items in a tuple:
  1. Convert the tuple to a list
  2. Make the changes as required to the list
  3. Convert the list back to a tuple

```
</> tupleExample1 = ("4061CEM", "Programming", "Algorithms")  
listExample1 = list(tupleExample1)
```

```
</> listExample1.append("Ian Cornelius")  
tupleExample1 = tuple(listExample1)
```

```
▶ [Before] tupleExample1 = ('4061CEM', 'Programming', 'Algorithms')  
[After] tupleExample1 = ('4061CEM', 'Programming', 'Algorithms', 'Ian Cornelius')
```

## MODIFYING A TUPLE (2)

### DELETING A TUPLE

- The entire tuple can be deleted and removed from the memory using the `del` keyword

```
</> tupleExample1 = ("4061CEM", "Programming", "Algorithms")
```

```
</> del tupleExample1
```

## MERGING A TUPLE

- The items of a tuple can be concatenated with the items of another tuple using the addition (“+”) operator

```
</> tupleExample1 = ("4061CEM", "Programming and Algorithms 1", "Dr Ian Cornelius")  
    tupleExample2 = ("4059CEM", "Legal and Ethical Foundations", "Mr Terry Richards")
```

```
</> mergedTupleExample1 = tupleExample1 + tupleExample2
```

```
▶ mergedTupleExample1 = ('4061CEM', 'Programming and Algorithms 1', 'Dr Ian Cornelius', '4059CEM', 'Legal and Ethical  
    Foundations', 'Mr Terry Richards')
```

## MULTIPLYING A TUPLE

- The items of a tuple can be multiplied to duplicate them, this can be achieved using the multiplying (“\*”) operator

```
</> tupleExample1 = ("4061CEM", "Programming", "Algorithms")
```

```
</> tupleExample2 = tupleExample1 * 3
```

```
▶ tupleExample2 = ('4061CEM', 'Programming', 'Algorithms', '4061CEM', 'Programming', 'Algorithms', '4061CEM',  
  'Programming', 'Algorithms')
```

## PACKING AND UNPACKING A TUPLE (1)

- Placing items within a tuple is known as **packing**
- The items of a tuple can be extracted to their own variable with a process known as **unpacking**

```
</> tupleExample1 = ("4061CEM", "Programming", "Algorithms")
```

```
</> (module_code, title1, title2) = tupleExample1
```

```
▶ module_code = 4061CEM  
  title1 = Programming  
  title2 = Algorithms
```



## PACKING AND UNPACKING A TUPLE (2)

- When unpacking you must match the number of variables to the number of items in the tuple
- If you have fewer variables than the number of items in the tuple; adding `*` to the variable name will assign remaining items in the tuple to a list

```
</> tupleExample1 = ("4061CEM", "Programming", "Algorithms")
```

```
</> (module_code, *title) = tupleExample1
```

```
▶ module_code = 4061CEM  
  title = ['Programming', 'Algorithms']
```

## PACKING AND UNPACKING A TUPLE (3)

- If you add the `*` to a variable that is not last, then Python will assign values to a list for that variable until the number of values left match the number of variables left

```
</> tupleExample1 = ("4061CEM", "Programming", "Algorithms", "1", "Dr Ian Cornelius", True)
```

```
</> (module_code, *title, leader, running) = tupleExample1
```

```
▶ module_code = 4061CEM  
  *title = ['Programming', 'Algorithms', '1']  
  leader = Dr Ian Cornelius  
  running = True
```

# COMMON BUILT-IN DATA TYPE METHODS

- Lists and Tuples have some common methods that are built-in directly to them:
  - count and index

## COUNT

- The `count()` method will return the number of times a specified value will occur in the list or tuple

```
</> listExample1 = ["4061CEM", "Programming", "Algorithms", "Algorithms", "algorithms"]
```

```
</> listExample1.count('Algorithms')
```

```
▶ listExample1.count('Algorithms') = 2
```

## INDEX

- The `index()` method will search the list for a specified value and return its index in the list or tuple

```
</> tupleExample1 = ("4061CEM", "Programming", "Algorithms")
```

```
</> tupleExample1.index("Programming")
```

```
▶ tupleExample1.index("Programming") = 1
```

# GOODBYE

- Questions?
  - Post them in the **Community Page** on Aula
- Contact Details:
  - Dr Ian Cornelius, [ab6459@coventry.ac.uk](mailto:ab6459@coventry.ac.uk)