MAPPING DATA TYPES

DR IAN CORNELIUS

HELLO

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

INTRODUCTION TO DICTIONARIES

- Dictionaries are used to store multiple items into a single variable
  - they are stored as a `key:value` pair
- 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 dictionary
  - **changeable**: the items of a dictionary are mutable (can be changed), added or removed
  - **no duplicates allowed**: dictionaries are unable to have the same key twice
- The size of a dictionary (or the number of items stored in a dictionary) can be determined using the `len()` function

CREATING A DICTIONARY

- Dictionaries are created by using a set of curly braces (`{}`)
- The `dict()` constructor can also be used to create a dictionary data type

```
dictExample1 = {
    "code": "4061CEM",
    "title": "Programming and Algorithms",
    "leader": "Ian Cornelius"
}
dictExample2 = dict({
    "code": "4061CEM",
    "title": "Programming and Algorithms",
    "leader": "Ian Cornelius"
})
dictExample3 = {"code": "4061CEM", "title": "Programming and Algorithms", "leader": "Ian Cornelius"}
```

DATA TYPES OF ITEMS IN A DICTIONARY (1)

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

```
dictExample1 = {
    "code": "4061CEM",
    "title": "Programming and Algorithms",
    "leader": "Ian Cornelius"
}
dictExample2 = {
    "code": 4061,
    "faculty": "CEM",
    "title": "Programming and Algorithms",
    "leader": "Ian Cornelius",
    "running": True
}
```

DATA TYPES OF ITEMS IN A DICTIONARY (2)

- Dictionaries can also store other dictionaries inside themselves

```
dictExample3 = {
    "code": 4061,
    "faculty": "CEM",
    "title": "Programming and Algorithms",
    "leader": "Ian Cornelius",
    "running": True,
    "resources": {
        "book1": {
            "author": "Heineman, G.T., Pollice, G. and Selkow, S.",
            "title": "Algorithms in a nutshell: A practical guide.",
            "year": "2016"
        }
    }
}
```

ACCESSING DICTIONARY ITEMS (1)

- The items in a dictionary can be accessed by referring to its key inside a set of square brackets (`[]`)

```
dictExample1 = {
        "code": "4061CEM",
        "title": "Programming and Algorithms",
        "leader": "Ian Cornelius"
    }
```

```
dictExample1["code"]
```

```
dictExample1[“code”] = 4061CEM
```

ACCESSING DICTIONARY ITEMS (2)

```
dictExample1 = {
    "code": 4061,
    "faculty": "CEM",
    "title": "Programming and Algorithms",
    "leader": "Ian Cornelius",
    "running": True,
    "resources": {
        "book1": {
            "author": "Heineman, G.T., Pollice, G. and Selkow, S.",
            "title": "Algorithms in a nutshell: A practical guide.",
            "year": "2016"
        }
    }
}
```

```
dictExample1["resources"]["book1"]["author"]
```

```
dictExample1[“resources”][“book1”][“author”] = Heineman, G.T., Pollice, G. and Selkow, S.
```

ACCESSING DICTIONARY ITEMS (3)

- The items in a dictionary can also be accessed by using the `get()` function, and a key

```
dictExample1 = {
    "code": "4061CEM",
    "title": "Programming and Algorithms",
    "leader": "Ian Cornelius"
}
```

```
dictExample1.get("code")
dictExample1.get("leader")
```

```
dictExample1.get(“code”) = 4061CEM
dictExample1.get(“leader”) = Ian Cornelius
```

ACCESSING DICTIONARY ITEMS (4)

LIST OF DICTIONARY KEYS

- A list of keys that are present in a dictionary can be retrieved using the `keys()` function
- The list of keys can be considered to be a view of the dictionary
    - therefore, any changes made to the dictionary will be rectified in the list of keys

```
dictExample1 = {
        "code": "4061CEM",
        "title": "Programming and Algorithms",
        "leader": "Ian Cornelius"
    }
```

```
dictExample1.keys()
```

```
dictExample1.keys() = dict_keys([‘code’, ‘title’, ‘leader’])
```

ACCESSING DICTIONARY ITEMS (5)

LIST OF DICTIONARY VALUES

- A list of values that are present in the dictionary can be retrieved using the `values()` function
- The list of values can be considered to be a view of the dictionary
  - therefore, any changes made to the dictionary will be rectified in the list of values

```
dictExample1 = {
        "code": "4061CEM",
        "title": "Programming and Algorithms",
        "leader": "Ian Cornelius"
    }
```

```
dictExample1.values()
```

```
dictExample1.values() = dict_values([‘4061CEM’, ‘Programming and Algorithms’, ‘Ian Cornelius’])
```

ACCESSING DICTIONARY ITEMS (6)

LIST OF DICTIONARY ITEMS

- A tuple of items that are present in the dictionary can be retrieved using the `items()` function
  - Each tuple returned is the `key` and `value` that is present in the dictionary
- The tuple of items can be considered to be a view of the dictionary
  - Therefore, any changes made to the dictionary will be rectified in the tuple of items

```
dictExample1 = {
        "code": "4061CEM",
        "title": "Programming and Algorithms",
        "leader": "Ian Cornelius"
    }
```

```
dictExample1.items()
```

```
dictExample1.items() = dict_items([(‘code’, ‘4061CEM’), (‘title’, ‘Programming and Algorithms’), (‘leader’, ‘Ian
Cornelius’)])
```

MODIFYING A DICTIONARY (1)

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

ADDING AN ITEM I

- Adding an item to the dictionary can be done using the `update()` function
  - when using this method, you must provide a `key:value` pair in the function

```
dictExample1 = {
        "code": "4061CEM",
        "title": "Programming and Algorithms",
        "leader": "Ian Cornelius"
    }
```

```
dictExample1.update({"running": True})
```

```
dictExample1.items() = dict_items([('code', '4061CEM'), ('title', 'Programming and Algorithms'), ('leader', 'Ian
Cornelius'), ('running', True)])
```

MODIFYING A DICTIONARY (2)

ADDING AN ITEM II

- The same `key:value` pair can be added using the square brackets (`[]`) notation

```
dictExample1 = {
      "code": "4061CEM",
      "title": "Programming and Algorithms",
      "leader": "Ian Cornelius"
  }
```

```
dictExample1["running"] = True
```

```
dictExample1.items() = dict_items([('code', '4061CEM'), ('title', 'Programming and Algorithms'), ('leader', 'Ian
Cornelius'), ('running', True)])
```

MODIFYING A DICTIONARY (4)

REMOVING ITEMS FROM A DICTIONARY I

- Items can be removed from a dictionary using the `pop()` function
  - this will remove the item from the list using its key

```
dictExample1 = {
        "code": "4061CEM",
        "title": "Programming and Algorithms",
        "leader": "Ian Cornelius"
    }
```

```
dictExample1.pop("title")
```

```
[Before] dictExample1.keys() = dict_keys([‘code’, ‘title’, ‘leader’])
[After] dictExample1.keys() = dict_keys([‘code’, ‘leader’])
```

MODIFYING A DICTIONARY (5)

REMOVING ITEMS FROM A DICTIONARY II

- An item can also be removed from a list by using the `popitem()` function
  - this will remove the last inserted item

```
dictExample1 = {
      "code": "4061CEM",
      "title": "Programming and Algorithms",
      "leader": "Ian Cornelius"
   }
```

```
dictExample1.popitem()
```

```
[Before] dictExample1.keys() = dict_keys([‘code’, ‘title’, ‘leader’])
[After] dictExample1.keys() = dict_keys([‘code’, ‘title’])
```

MODIFYING A DICTIONARY (6)

REMOVING ITEMS FROM A DICTIONARY III

- An item can also be removed from a dictionary by its key using the `del` keyword

```
dictExample1 = {
        "code": "4061CEM",
        "title": "Programming and Algorithms",
        "leader": "Ian Cornelius"
    }
```

```
del dictExample1["title"]
```

```
[Before] dictExample1.keys() = dict_keys(['code', 'title', 'leader'])
[After] dictExample1.keys() = dict_keys(['code', 'leader'])
```

MODIFYING A DICTIONARY (7)

CLEARING A DICTIONARY

- A dictionary 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 dictionary and leave it empty, symbolised by just the curly brackets (`{}`)

```
dictExample1 = {
    "code": "4061CEM",
    "title": "Programming and Algorithms",
    "leader": "Ian Cornelius"
}
```

```
dictExample1.clear()
```

```
[Before] dictExample1.keys() = dict_keys(['code', 'title', 'leader'])
[After] dictExample1 = {}
```

MODIFYING A DICTIONARY (8)

DELETING A DICTIONARY

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

```
dictExample1 = {
        "code": "4061CEM",
        "title": "Programming and Algorithms",
        "leader": "Ian Cornelius"
    }
```

```
del dictExample1
```

COPYING A DICTIONARY

- The method of copying a dictionary by using `dict2 = dict1` is incorrect
  - this method creates a reference to `dict1` and not an actual copy; therefore any changes made in `dict1` will occur in `dict2`
- The correct process of copying a list can be achieved by the `copy()` function or the `dict()` constructor itself

```
dictExample1 = {
    "code": "4061CEM",
    "title": "Programming and Algorithms",
    "leader": "Ian Cornelius"
}
```

```
copyDictExample1 = dictExample1.copy()
copyDictExample2 = dict(dictExample1)
```

GOODBYE

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